Microtec
Research Inc.

MCC68K

User's Guide

100134-010

TRADEMARKS

Apollo™, DOMAIN/OS™, and AEGIS™ are trademarks of Apollo Computer, Inc.

Flexible License Manager™ is a trademark of Highland Software, Inc.

High C® is a registered trademark of Metaware, Inc.

HP™ and HP-UX™ are trademarks of Hewlett-Packard Company.

IBM® is a registered trademark of International Business Machines, Inc.

ICE® is a registered trademark of Intel Corporation.

Microtec® and Paragon® are registered trademarks of Microtec Research, Inc.

MRI ASM™, MRI C™, MRI FORTRAN™, MRI Pascal™, and MRI XRAY™ are trademarks of Microtec Research, Inc.

MS-DOS™ is a trademark of Microsoft Corporation.

pSOS™ is a trademark of Software Components Group.

Sun™ is a trademark of Sun Microsystems, Inc.

Tektronix® is a registered trademark of Tektronix, Inc.

UNIX® is a registered trademark of AT&T.

VAX™, VMS™, and ULTRIX™ are trademarks of Digital Equipment Corporation.

VRTX™ is a trademark of Ready Systems, Inc.

XRAY68K™, XRAY86™, XRAY180™, XRAYZ80™, XRAY29K™, XRAY51™, XRAY88K™, XRAY960™, XRAYG32™, XRAYTX™, XRAY Debugger™, XRAY In-Circuit Debugger™, and XRAY In-Circuit Debugger Monitor™, XHM302™, XRAYH83™, and XRAYH85™ are trademarks of Microtec Research, Inc.

Zilog® is a registered trademark of Zilog Corporation.

302bug™ is a trademark of Motorola, Inc.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 of DOD Regulation 27-401, or comparable clause, as such may be amended or renumbered from time to time.

MICROTEC RESEARCH, INCORPORATED
2350 MISSION COLLEGE BLVD.
SANTA CLARA, CA 95054

© Copyright 1987, 1988, 1989, 1990, 1991 Microtec Research, Inc. All rights reserved. No part of this publication may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical or otherwise, without prior written permission of Microtec Research, Inc.

REV.	REVISION HISTORY	DATE	APPD.
-001	Original issue.	8/88	T.P.
-002	Corrections.	11/88	T.P.
-003	Revised to meet ANSI C standards.	4/89	T.P
-004	(Non-ANSI) IBM-PC user's instructions added and produced at 80% for new packaging.	12/89	T.P.
-005	ANSI MCC68K. Includes IBM-PC/MS-DOS, UNIX, and reduced to 80% for new packaging.	12/89	T.P.
-006	Added user's guide instructions for VAX/VMS, Apollo DOMAIN/IX, and Motorola Delta Systems UNIX/System V systems.	2/90	T.P.
-007	Added support for VAX/ULTRIX and HP-UX.	4/90	J.A.
-008	Manual revision to support software Version 4.1.	6/90	T.P.
-009	Manual production upgraded from Framemaker version 1.3 to 2.0.	9/90	T.P.
-010	Manual revised to support version 4.2 software.	9/91	L.W.

Preface

Overview

This guide describes how to use the Microtec Research MCC68K C Compiler for the Motorola 68000 family of microprocessors. MCC68K operates on the following host computers and operating systems:

- Apollo DOMAIN/OS
- DECstation/ULTRIX
- HP 9000/HP-UX
- IBM-PC/AT or compatible computer using PC-DOS or MS-DOS
- SCO UNIX System V/386
- Sun-3/SunOS or Sun-4/SunOS
- VAX/VMS

About this Documentation Set

The documentation set describing the Microtec Research MCC68K Compiler consists of this *User's Guide*, an *Installation Guide*, and a *Reference Manual*.

1. The *User's Guide* provides host computer and operating system-specific descriptions of how to invoke the compiler.

2. The *Reference Manual* describes the MCC68K implementation of the C programming language and the internal operation of the compiler. Error and warning messages are described.

3. The *Installation Guide* contains host computer and operating system-specific directions for installing the MCC68K package.

About this Guide

The *Microtec Research MCC68K User's Guide* consists of the following chapters:

- Chapter 1, *Introduction*, describes the components of the MCC68K C Compiler.

- Chapter 2, *VAX/VMS User's Guide*, describes how to use the compiler on the VMS operating system.

- Chapter 3, *UNIX/DOS User's Guide*, describes how to use the compiler on SunOS, UNIX and UNIX-like operating systems, and IBM-PC/DOS.

Refer to the appropriate chapter for your particular host computer and operating system, and disregard the other chapters.

Related Publications

This documentation assumes that you are familiar with programming and with the C programming language.

For general information about C, refer to the following publications:

- *The C Programming Language*, 2nd ed., by Brian W. Kernighan and Dennis M. Ritchie, Prentice-Hall, Inc., 1988.

  This book contains an introduction to C with examples.

- *C — A Reference Manual*, 3rd ed., by Samuel P. Harbison and Guy L. Steele, Jr., Prentice-Hall Inc., 1991.

  This book contains a complete discussion of both ANSI C and traditional "K & R" C. It includes all library functions.

For additional information about MCC68K, refer to the following publication:

- *Getting the Most Out of Your Microtec Research MCC68K ANSI C Compiler*, Microtec Research, Inc.

  This document gives you tips to help you get the most out of your MCC68K Compiler.

For information about other Microtec Research 68K toolkit components, refer to the following publications:

- *ASM68K User's Guide, Reference Manual*, and *Installation Guide*, Microtec Research, Inc.

  These manuals describe how to use the ASM68K Assembler, the LNK68K Linker, and the LIB68K Librarian.

- *XRAY68K User's Guide, Reference Manual*, and *Installation Guide*, Microtec Research, Inc.

  These manuals describe how to use the XRAY68K Debugger.

Notational Conventions

The *Microtec Research MCC68K User's Guide* uses the notational conventions shown in Table P-1 (unless otherwise noted).

Table P-1. Notational Conventions

Symbol	Name	Usage
{ }	Curly Braces	Encloses a list from which you must choose an item.
[ ]	Square Brackets	Encloses items that are optional.
...	Ellipsis	Indicates that you may repeat the preceding item zero or more times.
|	Vertical Bar	Separates alternative items in a list.
	Punctuation	Punctuation such as commas (,) and colons (:) must be entered as shown.
	`Typewriter Font`	Represents user input in interactive examples.
	Italics	Represents a descriptive item that should be replaced with an actual item.

Questions and Suggestions

To help us respond to questions or suggestions regarding the MCC68K Compiler, we have provided two response forms in the back of this documentation.

The first form is a Reader's Response sheet, which is used to help us correct and improve our documentation. We always appreciate your comments, and by completing this form, you can participate directly in the revisions of our publications. The Reader's Response sheet is directed to Technical Publications.

The second form is a Software Performance Report that should be completed if you encounter any errors or problems with Microtec Research software. This report is directed to Technical Support.

Contents

Preface

Overview v
About this Documentation Set v
About this Guide v
Related Publications vi
Notational Conventions vii
Questions and Suggestions vii

1 Introduction

Components of the Microtec Research MCC68K Package 1-1
MCC68K Compiler 1-1
ASM68K Assembler 1-1
LNK68K Linker 1-2
LIB68K Object Module Librarian 1-2
XRAY68K Debugger (optional) 1-2
Data Flow 1-3

2 VAX / VMS User's Guide

Overview 2-1
Invoking the Compiler 2-1
Command Line Syntax 2-1
File Name Extensions 2-2
Input and Output File Locations 2-2
Command Line Option Summary 2-3
Command Line Options 2-9
Choose Structure Alignment 2-9
Set ANSI-Compliant Mode 2-10
Produce Assembler Source File 2-10
Display Banner 2-10
Generate CLR Instruction 2-10
Generate Position-Independent Code and Data 2-11
Choose Address Mode for Sections 2-12
Produce Code for Specified Processor 2-14
Produce Debug Information 2-14
Define a Preprocessor Macro Name 2-16
Generate Floating-Point Processor Instructions 2-16
Generate LINK/UNLK in Prologue/Epilogue 2-17
Support HP 64000 2-17
Change Insert Character 2-17

Add Search Path for Include Files 2-17
Return from Interrupt with RTS 2-18
Write List File to Standard Output 2-19
Enable Microtec Research Extensions 2-19
Optimize Code 2-20
Specify Format of Output Files 2-22
Execute C Preprocessor Only 2-23
Modify Naming Conventions 2-23
Stop Compilation Due to Errors 2-24
Name the Sections 2-24
Suppress Diagnostic Messages 2-25
Check Program Syntax 2-26
Generate Trace Information 2-26
Truncate Name 2-27
Undefine a Preprocessor Macro Name 2-27
Treat char as Unsigned 2-27
Initializing Uninitialized Global Data 2-27
Command Line Examples 2-29
Invoking Other MCC68K Components 2-30
ASM68K Invocation 2-30
LNK68K Invocation 2-31
Sample Program 2-32
Generating an Executable Program 2-32
Running on a Target System 2-33
Libraries 2-34
Distribution Files 2-34
The C Library 2-35
UNIX System Functions 2-35
Start-Up Routine 2-36

3 UNIX / DOS User's Guide

Overview 3-1
Invoking the Compilation Driver 3-1
Command Line Syntax 3-1
Default File Name Extensions 3-2
Input File Name Extensions 3-2
Output File Name Extensions 3-3
Input and Output File Locations 3-3
Environment Variables 3-3
Considerations for DOS 3-5
Compiling Large Modules (DOS only) 3-6
Use Extended Memory Compiler 3-6
Preprocess Separately 3-6
Disable ANSI Features to Avoid Prototypes in Standard Include Files 3-7

Command Line Option Summary 3-8
Command Line Options 3-14
Set ANSI-Compliant Mode 3-14
Choose Address Mode for Sections 3-15
Invoke Extended Memory Compiler (PC only) 3-16
Save Comments 3-16
Produce an Object File (driver option) 3-16
Define a Preprocessor Macro Name 3-16
Read Options from File (driver option) 3-17
Invoke C Preprocessor Only (driver option) 3-17
Pass Command File to Linker (driver option) 3-17
Specify Format of Output Files 3-17
Generate Floating-Point Processor Instructions 3-18
Produce Debug Information 3-18
Support HP 64000 3-20
Add Search Path for Non-Standard Include Files 3-20
Add Search Path for Standard Include Files 3-21
Generate Control Prologue/Epilogue Protocol and Code Variations 3-21
Write List File to Standard Output 3-23
Generate Position-Independent Code and Data 3-23
Name the Sections 3-24
Optimize Code 3-25
Name the Output File (driver option) 3-27
Execute C Preprocessor Only (driver option) 3-28
Produce Code for Specified Processor 3-28
Suppress Diagnostic Messages 3-29
Produce Assembler Source File (driver option) 3-29
Undefine a Preprocessor Macro Name 3-29
Modify Naming Conventions 3-29
Control Verbose Output Information 3-30
Pass Options to Tools (driver option) 3-31
Initializing Uninitialized Global Data 3-31
Enable Microtec Research Extensions 3-31
Check Program Syntax (driver option) 3-32
Choose Structure Alignment 3-32
Command Line Examples 3-34
Invoking the Linker 3-35
Libraries 3-36
Distribution Files 3-36
The C Library 3-37
UNIX System Functions 3-37
Start-Up Routine 3-38

Appendix A: UNIX / DOS — VMS Equivalences A-1

Figures

Figure 1-1. Data Flow Through the Compiler 1-4

Tables

Table 2-1. VMS Default File Name Extensions 2-2
Table 2-2. Command Line Option Summary (VMS) 2-3
Table 2-3. Allocation of Code and Data (VMS) 2-12
Table 2-4. Library Use (VMS) 2-34
Table 2-5. UNIX Functions Used by the MCC68K Library (VMS) 2-36
Table 3-1. UNIX/DOS Default Input File Name Extensions 3-2
Table 3-2. UNIX/DOS Default Output File Name Extensions 3-3
Table 3-3. Environment Variables and UNIX/DOS Defaults 3-4
Table 3-4. Command Line Option Summary (UNIX/DOS) 3-8
Table 3-5. Allocation of Code and Data (UNIX/DOS) 3-15
Table 3-6. Library Use (UNIX/DOS) 3-36
Table 3-7. UNIX Functions Used by the MCC68K Library (UNIX/DOS) 3-38
Table A-1. UNIX/DOS — VMS Equivalent Options A-1

Introduction 1

Components of the Microtec Research MCC68K Package

The MCC68K C Compiler is part of the software development toolkit for the 68K family of microprocessors available from Microtec Research. Other components include a relocatable cross assembler, a linker, and an object module librarian. The popular XRAY Debugger is optionally available. Programs are delivered ready to run on your host computer.

The components of the Microtec Research 68K toolkit are described below.

MCC68K Compiler

The MCC68K optimizing cross compiler converts both ANSI C and traditional C source programs into tight, efficient assembly language code for ASM68K.

The compiler has options which you can control. To name only a few examples, you can create ROMable programs, generate position-independent code and data, support register-relative data addressing, suppress warnings, and redirect listings. The MCC68K Compiler contains an ANSI C preprocessor that you can run independently. The C compiler can produce information necessary for debugging with the XRAY68K Debugger.

ASM68K Assembler

The Microtec Research ASM68K Assembler converts assembly language programs into relocatable object code. Object modules are suitable for linking with other modules or with libraries of modules.

The assembler accepts source program statements that are syntactically compatible with those accepted by Motorola 68000 family assemblers. The ASM68K Assembler also processes macros and conditional assembly statements. The assembler generates a cross-reference table as well as a standard symbol table. Code and data are placed in multiple named sections.

After assembly, the linker links the object modules, which are then suitable for download and execution on any Motorola 68000, 68010, 68020, 68030, 68040, or CPU32 family microprocessor.

LNK68K Linker

The Microtec Research LNK68K Linker combines relocatable object modules into a single absolute object module. You can optionally generate object modules in one of three different formats: Motorola S-record, HP 64000 OMF, or MRI-IEEE. Additionally, the linker can combine multiple relocatable object modules into a single relocatable module, which subsequently can be relinked with other modules. This feature is referred to as incremental linking.

If one of the input files is a library of object modules, the linker automatically loads all modules from the library that are referenced by externals in the other named object modules. The linker produces a link map that shows the final location of all modules and sections, and the final absolute values of all symbols. A cross-reference listing shows which modules refer to each global symbol.

The linker reports unresolved external symbols as well as errors that occur at link time, and prints these messages on the link map or on the terminal.

LIB68K Object Module Librarian

The Microtec Research LIB68K Librarian is a library maintenance program that can build program libraries. A program library is a collection of relocatable object modules that reside in one file.

Libraries let you automatically load frequently used object modules without concern for the specific names and characteristics of the modules. The LIB68K librarian program lets you format and organize library files that will subsequently be used by the LNK68K Linker.

XRAY68K Debugger (optional)

The Microtec Research XRAY68K Debugger lets you monitor and control the execution of programs at both the high-level C and assembly level. You can examine or modify the value of program variables using the same source-level terms, definitions, and structures that were defined in the original source code. To display information, XRAY68K employs a window-oriented user interface that divides debugging information into meaningful sections.

The debugger is interactive and gives you complete control of the program by directing an execution environment such as a simulator, in-circuit emulator, single board computer, or target monitor. It executes your program while allowing you to access variables, procedures, source line addresses, and other program entities.

XRAY68K's powerful command language allows simple and complex breakpoint setting, single stepping, and continuous variable monitoring. Input and output can be directed to/from files, buffers, or viewports. In addition, a sophisticated macro facility allows complex command sequences to be associated with events such as the execution of a specific statement or the accessing of a specific data location. These features allow you to isolate errors and patch your source code.

Data Flow

Figure 1-1 illustrates the components of the Microtec Research MCC68K C compiler package and the flow of data among them.

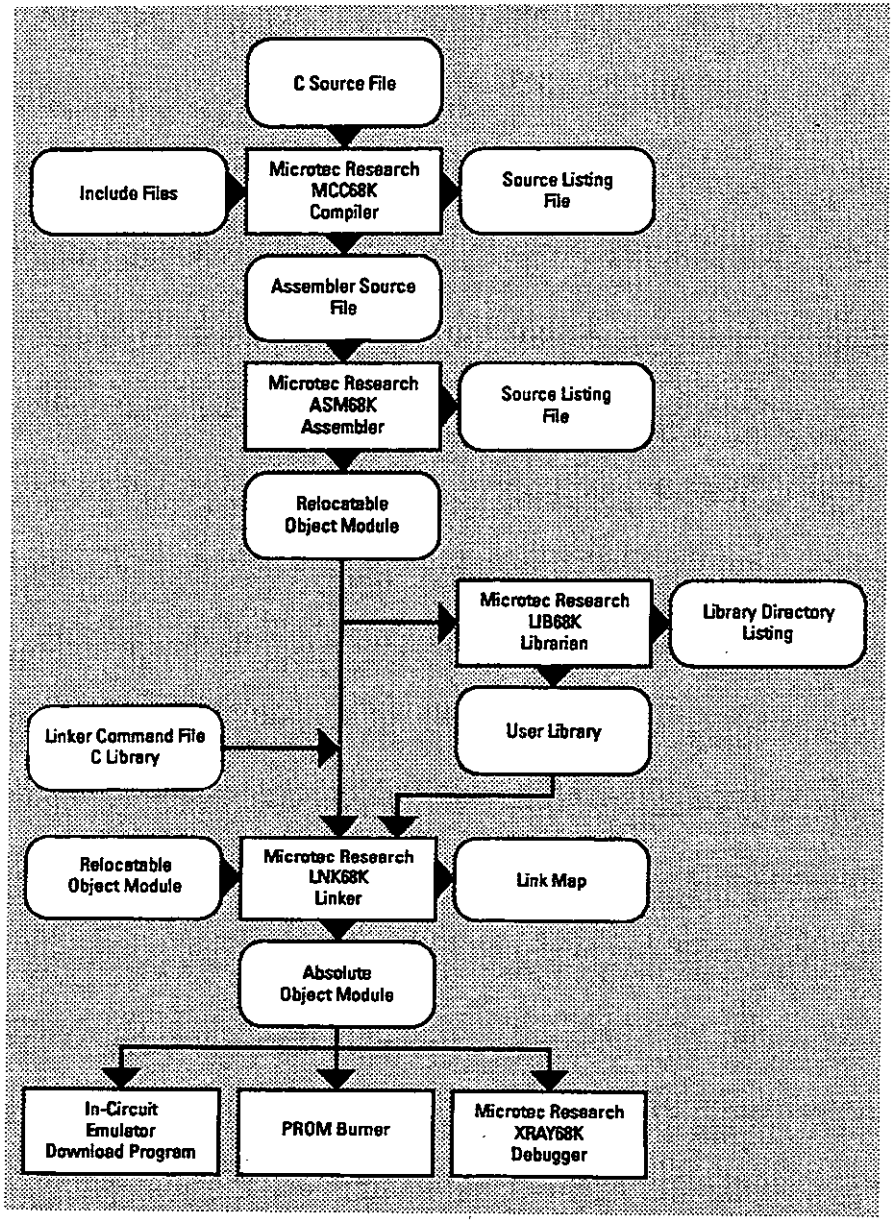

Figure 1-1. Data Flow Through the Compiler

VAX / VMS User's Guide 2

Overview

This chapter describes how to use the Microtec Research MCC68K C Compiler on a VAX host computer running the VMS operating system.

The chapter includes:

- A description of how to invoke the compiler
- Descriptions of command line options and defaults
- Quick reference tables summarizing the command line options
- Example command lines that make use of various options
- An example of how to compile, assemble, and link a sample program

Invoking the Compiler

This section describes the compiler's invocation command, default input and output file name extensions, and default file locations.

Command Line Syntax

The following command invokes the compiler:

`c68k [/option]...` *source_filename* `[/option]...`

Description:

`c68k` — The name of the C compiler.

/option — Any of the options described later in the section *Command Line Options*. Each option must be preceded by a slash (/). Options have the following syntax:

/option`[=`*suboption*`]`

where *suboption* qualifies the *option*.

Options can be placed after the compiler name or after the *source_filename*. If no options are specified, the default options are used.

source_filename The name of a C program source file. By default, .c is the assumed source file name extension. File names can be fully qualified according to VMS syntax with a node name, device name, directory name, file name extension, and version number.

File Name Extensions

The compiler uses the file name extensions listed in Table 2-1.

Table 2-1. VMS Default File Name Extensions

File	Extension
C intermediate preprocessed file	.i
C source file (from user)	.c
Assembly language file	.src
Relocatable object file	.obj
Compiler listing file	.lis
Executable file	.abs

Input and Output File Locations

If a node, device, or directory is not specified for an input file, the current defaults are used.

If output file names are not specified, or if node, device, or directory names are not specified with output file names, the output files are placed in the current directory.

Command Line Option Summary

Table 2-2 summarizes the positive form of MCC68K command line options.

Table 2-2. Command Line Option Summary (VMS)

Option/Suboption	Meaning
/addresses=long	(68020/030/040/CPU32 modes) Directs the compiler to use 32-bit displacements for A*n*- or PC-relative addressing.
/addresses=short	Directs the compiler to use 16-bit displacements for A*n* or PC-relative addressing.
/align=2	Uses even-alignment for structure members larger than char.
/align=4	Uses quad-alignment for structure members larger than short.
/ansi	Sets ANSI-compliant mode.
/asm[=*name*]	Names the output file.
/banner	Displays compiler banner.
/clr	Directs the compiler to use the CLR instruction.
/code_addresses=absolute	Directs the compiler to use absolute addressing for all code references.
/code_addresses=pcrelative	Directs the compiler to use PC-relative addressing for all code references.
/const_addr_as=code	References items in the const section according to the /code_addresses option.
/const_addr_as=data	References items in the const section according to the /data_addresses option.
/cpu=*processor*	Produces code for a specified processor.
/data_addresses=absolute	Directs the compiler to use absolute addressing for all data references.

(cont.)

Table 2-2. Command Line Option Summary (cont.)

Option/Suboption	Meaning
/data_addresses=a*n*relative	Directs the compiler to use A*n*-relative addressing for all static data references.
/data_addresses=pcrelative	Directs the compiler to use PC-relative addressing for all static data references.
/debug	Generates debug information.
/debug=fullfilename	Generates fully-qualified path names for input files along with debug information.
/debug=lines	Generates line number labels.
/debug=multi_stmt	Generates debug information for multi-statement lines along with other debug information.
/debug=restricted	Generates restricted debug information.
/define=*name*[=*string*]	Defines a preprocessor symbol *name* to have the value *string*.
/fpu	Generates code that uses instructions provided by the floating-point coprocessor.
/frames	Generates LINK/UNLK instructions in the function prologue and epilogue.
/hp64000	Generates code for HP 64000 Series Development Systems.
/initvars_addr_as=code	References items in the initialized data section according to the /code_addresses option.
/initvars_addr_as=data	References items in the initialized data section according to the /data_addresses option.
/insert_char=*char*	Changes the default insert character in assembler inlining from a back quote (`) to *char*.

(cont.)

Table 2-2. Command Line Option Summary (cont.)

Option/Suboption	Meaning
/ipath=*dir*	Searches the specified path to locate user #include files.
/ireturn=exception	Uses RTE to return from an interrupt function.
/ireturn=subroutine	Uses RTS to return from an interrupt function.
/list[=*filename*]	Generates a source listing with errors.
/literals_addr_as=code	References items in compiler-generated literals according to the /code_addresses option.
/literals_addr_as=data	References items in compiler-generated literals according to the /data_addresses option.
/mri_extensions	Enables Microtec Research extensions to the C language.
/optimize=combine_pops	Does not pop the stack after each function call.
/optimize=globalflow	Invokes global flow optimizer.
/optimize=inline	Enables inlining.
/optimize=local	Enables local optimizations.
/optimize=register	Allocates heavily used variables to registers.
/optimize=reorder	Enables instruction reordering optimizations.
/optimize=singleret	Generates only one exit (return) from a function.
/optimize=size	Optimizes in favor of code size rather than execution time.

(cont.)

Table 2-2. Command Line Option Summary (cont.)

Option/Suboption	Meaning
/optimize=stablemem	Optimizes global variables if /optimize=globalflow is also specified.
/optimize=time	Optimizes in favor of execution time rather than code size.
/pagelength=*number*	Sets the page length of the listing file to *number*.
/pp	Creates preprocessor output but does not compile. It also creates a .I file.
/pp=comment	Lists comments in the preprocessor output.
/prepend=dot	Prepends a dot to all symbol names.
/prepend=underscore	Prepends an underscore to all symbol names.
/quit=*number*	Stops compilation if more than *number* errors occur.
/rename=(code=*name*)	Sets the code section name to *name*.
/rename=(const=*name*)	Sets the const section name to *name*.
/rename=(initvars=*name*)	Sets the initialized data section name to *name*.
/rename=(literals=*name*)	Sets the compiler-generated literals section name to *name*.
/rename=(module=*name*)	Sets the module name to *name*.
/rename=(strings=*name*)	Sets the strings section name to *name*.
/rename=(zerovars=*name*)	Sets the uninitialized static data section name to *name*.
/reserve=a*n*	Directs the compiler to reserve an A*n* register.
/show=include	Lists the contents of each include file.

(cont.)

Table 2-2. Command Line Option Summary (cont.)

Option/Suboption	Meaning
/show=source	Intermixes assembly language and source code in the output file.
/spath=*dir*	Searches the specified path to locate standard #include files.
/strings_addr_as=code	References items in the strings section according to the /code_addresses option.
/strings_addr_as=data	References items in the strings section according to the /data_addresses option.
/suppress	Suppresses diagnostic messages.
/suppress=error	Suppresses error, warning, and informational messages.
/suppress=informational	Suppresses informational messages.
/suppress=summary	Suppresses summary message.
/suppress=warning	Suppresses warning and informational messages.
/syntax_only	Checks source syntax but does not compile.
/title=*string*	Specifies title for the listing file.
/trace	Generates code to tag a function's exit and entry points.
/truncate_identifiers=*number*	Truncates the length of program identifiers to *number*.
/undefine=*name*	Undefines the defined preprocessor symbol.
/unsignedchar	Treats plain chars as unsigned.
/weakextern=common	Treats uninitialized global variables as "weak externals" or "C common".

(cont.)

Table 2-2. Command Line Option Summary (cont.)

Option/Suboption	Meaning
/weakextern=initzero	Allocates space for global variables that have not been explicitly initialized, and initializes them to zero.
/weakextern=public	Allocates space for global variables that have not been explicitly initialized.

Command Line Options

Command line options specify the names of output files and turn control switches on and off.

You can use upper-case or lower-case characters (or any combination) for option names. Each option can be abbreviated by using the minimum number of characters needed to distinguish it from any other option.

Some options have a negative form. The negative form disables or "turns off" the option. You can specify the negative form by preceding the option or its abbreviation with no.

If conflicting options are specified in the same command line, the rightmost specified option takes effect. For example, if a command line includes an option followed by the negative form, the negative form will take effect.

Use parentheses to specify multiple settings for the same option. For example, /optimize=(size,nolocal) will optimize code for size with no local optimizations. However, if you had entered /optimize=size /optimize=nolocal, only the last optimize parameter is used; the code is not optimized for size.

The following pages describe the command line options for the MCC68K Compiler on the VAX/VMS operating system.

Choose Structure Alignment

/align=2 (default for /cpu=68000, /cpu=68010, or /cpu=cpu32)
/align=4 (default for /cpu=68020, /cpu=68030, or /cpu=68040)

The default alignment of structured members depends on the processor selected with the /cpu option. In the 68000/010/CPU32 mode, non-char members will be even-aligned. In 68020/030/040 mode, 4-byte members will be quad-aligned and 2-byte members will be even-aligned.

With the /align=2 option, non-char members will be even-aligned, independent of the processor selected.

With the /align=4 option, 4-byte members will be quad-aligned and 2-byte members will be even-aligned, independent of the processor selected.

The alignment options do not affect packed structures. For information on packed structures, refer to the section titled *Packed Structures* in the *Reference Manual.*

Set ANSI-Compliant Mode

/ansi (default)
/noansi

With the /ansi option, additional features provided by ANSI C are enabled.

With the /noansi option, the compiler will accept pre-ANSI C features by disabling any ANSI feature that may invalidate it. For example, if /noansi is used, const will not be recognized as a keyword.

The /noansi option does not guarantee compatibility with earlier versions of the MCC68K Compiler or with other pre-ANSI compilers.

Produce Assembler Source File

/asm[=*name*]

With the /asm[=*name*] option, the compiler directs assembly language output to the specified file. If this option is not specified or if *name* is omitted, the source file name will be used with the extension replaced by .src.

Display Banner

/banner
/nobanner (default)

With the /banner option, the compiler banner displays the compiler version number and the copyright notice.

Generate CLR Instruction

/clr (default)
/noclr

With the /clr option, the compiler will use the CLR instruction for assigning zero to register and memory variables.

With the /noclr option, the compiler will not use the CLR instruction for assigning zero to non-register variables. On some 68000 processors, the CLR instruction does a read of memory before writing a zero to the location. This may cause problems for memory-mapped I/O ports.

Generate Position-Independent Code and Data

/addresses=long (68020/030/040/CPU32 modes) With the /addresses=long option, the compiler uses 32-bit displacements for A*n*- or PC-relative addressing.

/addresses=short (default)
With the /addresses=short option, the compiler uses 16-bit displacements for A*n*- or PC-relative addressing.

/code_addresses=absolute (default)
With the /code_addresses=absolute option, the compiler uses absolute addressing for all code references.

/code_addresses=pcrelative
With the /code_addresses=pcrelative option, the compiler uses the PC-relative addressing mode for all code references. This results in code that is position independent. The preprocessor symbol _PIC will be predefined.

/data_addresses=absolute (default)
With the /data_addresses=absolute option, the compiler uses absolute addressing for all data references. If you use this option, link with one of the absolute (ab) libraries.

/data_addresses=a*n*relative
With the /data_addresses=a*n*relative option, the compiler uses A*n*-relative addressing for all references to static data. This results in data that is position independent. The preprocessor symbol _PID will be predefined.

The register number *n* must be in the range 2 to 6. If you use the A5-relative addressing option, link with one of the A5-relative (a5) libraries. For data generated relative to other valid registers, you must build a corresponding library. See the *MCC68K Reference Manual* for information on modifying the default linker command file and for more information about register-relative global data references.

/data_addresses=pcrelative
With the /data_addresses=pcrelative option, the compiler uses PC-relative addressing for all references to static data. This results in data that is position independent. The preprocessor symbol _PID will be predefined.

If you use this option, link with one of the PC-relative (pc) libraries.

/reserve=a*n* With the /reserve=a*n* option, the compiler reserves a single A*n* register. The compiler will not generate any code that uses the reserved register. The register number *n* must be in the range 2 to 6.

If you enter /reserve=A*n* more than once on the command line, the last occurrence of /reserve=A*n* will override previous occurrences. For example, if you enter:

```
/reserve=A2 /reserve=A3
```

the `/reserve=A2` will be discarded and the compiler will not generate any code that uses the A3 register.

Note

The options /code_addresses=absolute and /code_addresses=pcrelative are mutually exclusive.

The options /data_addresses=absolute, /data_addresses=pcrelative, and /data_addresses=a*n*relative are mutually exclusive.

The options /data_addresses=a*n*relative and /reserve=a*n* are mutually exclusive.

Choose Address Mode for Sections

The compiler and linker allocate code and data to sections listed in Table 2-3.

Table 2-3. Allocation of Code and Data (VMS)

Contents	Section Name
code	code
uninitialized data	zerovars
initialized data	vars
const vars	const
strings	strings
compiler-generated literals	literals
compiler-generated tag data	tags (/trace option only)

Items in code are referenced according to the address mode chosen with the /code_addresses=absolute or /code_addresses=pcrelative option. Items in zerovars and tags are referenced according to the address mode chosen with the /data_addresses=absolute, /data_addresses=pcrelative, or /data_addresses=*a*nrelative option.

You can reference items in the remaining sections (vars, const, strings, literals) according to the address mode chosen with the options /code_addresses or /data_addresses, depending on where the sections will be placed at link time. If a section is to be placed in ROM (with code), its items should be referenced according to the /code_addresses option. If a section is to be placed in RAM (with zerovars and vars), its items should be referenced according to the option /data_addresses.

/const_addr_as=code (default)
/const_addr_as=data

With the /const_addr_as=code option, items in const will be referenced according to the /code_addresses option.

With the /const_addr_as=data option, items in const will be referenced according to the /data_addresses option.

/initvars_addr_as=code
/initvars_addr_as=data (default)

With the /initvars_addr_as=code option, items in vars will be referenced according to the /code_addresses option.

With the /initvars_addr_as=data option, items in vars will be referenced according to the /data_addresses option.

/literals_addr_as=code (default)
/literals_addr_as=data

With the /literals_addr_as=code option, items in literals will be referenced according to the /code_addresses option.

With the /literals_addr_as=data option, items in literals will be referenced according to the /data_addresses option.

/strings_addr_as=code (default)
/strings_addr_as=data

With the /strings_addr_as=code option, items in strings will be referenced according to the /code_addresses option.

With the /strings_addr_as=data option, items in strings will be referenced according to the /data_addresses option.

Produce Code for Specified Processor

/cpu=*processor*

With the /cpu=*processor* option, the compiler produces code for the specified processor. The possible values for processor are 68000 (default), 68010, 68020, 68030, 68040, and cpu32.

If you try to use the /cpu=68332 option, you will get a warning message indicating that this option is obsolete; the compiler will then default to /cpu=cpu32.

The /cpu=*processor* option affects the alignment of global data, local data, and structs. Modules which share structs must be compiled for the same processor. See the *Reference Manual* for more information on alignment.

This option causes one of the following preprocessor symbols to be predefined: _68000, _68010, _68020, _68030, _68040, _CPU32.

If you use the /cpu=68020, /cpu=68030, /cpu=68040, or /cpu=cpu32 option, link with one of the 020 libraries.

Produce Debug Information

/debug
/nodebug (default)

With the /debug option, the compiler generates line number, and variable and tracing information. This information is in Microtec Research format, which can be used by the XRAY68K Debugger. With /debug, you must use the Microtec Research ASM68K Assembler and LNK68K Linker.

/debug=fullfilename
/debug=nofullfilename (default)

The /debug=fullfilename option is a superset of the /debug option.

With the /debug=fullfilename option, the compiler generates fully-qualified path names for input files in addition to debug information. (A fully-qualified file name has a full path name even if one was not specified for the file name given on the command line.) The fully-qualified file name lets XRAY68K find the source file (.c) even if XRAY68K is not invoked from the directory in which the source file resides.

/debug=lines
/debug=nolines (default)

The /debug=lines option and the /debug option are mutually exclusive.

With the /debug=lines option, the compiler generates line number labels of the form LL*nn*, where *nn* is the line number. These line number labels will not be recognized by XRAY68K, but may be used in other debugging environments.

/debug=multi_stmt (default)
/debug=nomulti_stmt

The /debug=multi_stmt option is a superset of the /debug option.

With the /debug=multi_stmt option, the compiler generates debug information for "stepping" on individual statements on lines containing more than one statement.

This option requires the XRAY68K Debugger, version 2.2 or later. If it is used with earlier versions of XRAY68K, it may be necessary to do multiple "step" commands to get past a line of source code containing more than one statement.

/debug=restricted
/debug=norestricted (default)

The /debug=restricted option and the /debug option are mutually exclusive.

With the /debug=restricted option, the compiler generates "restricted" debug information. For each function, information concerning its frame type, push mask, and start and end addresses will be generated. No debug information will be generated for line numbers or symbols.

This option will typically be used when compiling modules that have already been debugged. This option lets the XRAY68K Debugger economically provide call traceback information for the functions in those modules. This functionality is present in XRAY68K, version 2.2 and subsequent versions.

Define a Preprocessor Macro Name

/define=*name*[=*string*]

With the /define=*name*[=*string*] option, you can define the value of one or more preprocessor macros. If *string* is not specified, the preprocessor macro has the value 1. This is equivalent to putting #define *name* 1 at the top of the source file.

VMS is case-insensitive. To preserve upper- and lower-case distinction, place the macro name in double quotes.

Separate multiple definitions by commas and enclose the definition set in parentheses. For example:

```
/def=("abc", ABC)
```

is equivalent to including the preprocessor directives #define abc 1 and #define ABC 1.

You can also assign values to individual macros. For example:

```
/def=(A=10, B=20)
```

See the *MCC68K Reference Manual* for more information about /define and preprocessor directives.

The companion /undefine option is always processed before the /define command line option, regardless of the order in which they appear on the command line.

Generate Floating-Point Processor Instructions

/fpu
/nofpu (default)

With the /fpu option, the compiler generates code that utilizes the instructions provided by the 68881 floating-point coprocessor. The preprocessor symbol _FPU will be predefined.

If /cpu=68040 has been specified, the /fpu option will be ignored.

With /nofpu, all floating-point operations are implemented by making calls to the run-time library.

Generate LINK/UNLK in Prologue/Epilogue

/frames
/noframes (default) With the /frames option, all functions will be "framed". A framed function has a LINK/UNLK pair of instructions in the function prologue and epilogue.

With the /noframes option, the compiler will generate a frame for a function only if it is needed.

Support HP 64000

/hp64000
/nohp64000 (default) The /hp64000 option should be used in conjunction with the /debug option.

With the /hp64000 option, the compiler generates code suitable for use on Hewlett-Packard HP 64000 Series Development Systems. The code includes two special labels, R*label* and E*label*, that the HP 64000 uses to identify each function's exit and end point for debugging and timing purposes. R*label* represents the function's return point, and E*label* represents the function's end address. The *label* portion of the symbol name is the name of the function.

Change Insert Character

/insert_char=*char* With the /insert_char=*char* option, the default insert character in assembler inlining is changed from a back quote (`) to *char*.

Add Search Path for Include Files

/ipath=*dir* With the /ipath=*dir* option, you can specify a search path to be scanned to locate user #include files. If an include file name is enclosed in double quotes (" "), the compiler will search for the include file:

1. in the directory containing the source file.
2. in the directories specified with the /ipath option.

   Enter directories as full directory names enclosed by square brackets. Specify multiple directories by

separating directory names with a comma and enclosing the list in parentheses. For example:

```
/ipath=([sys1.us1],[sys2.us2])
```

will cause the compiler to search for include files in the directory `[sys1.us1]` first, followed by the directory `[sys2.us2]`.

3. in the directories specified with the /spath option.
4. in the standard include file directory **c68k$library:**.

See the /spath option for information on include file names enclosed in angle brackets (< >).

/spath=*dir* With the /spath=*dir* option, you can specify a search path to be scanned to locate standard #include files. When an include file name in your source file is enclosed in angle brackets (< >), the compiler will search for the include file:

1. in the directories specified by the /spath option.

   Enter directories as full directory names enclosed by square brackets. Specify multiple directories by separating directory names with a comma and enclosing the list in parentheses. For example:

   ```
   /spath=([sys1.us1],[sys1.us2])
   ```

   will cause the compiler to search for include files in the directory `[sys1.us1]` first, followed by the directory `[sys1.us2]`.

2. in the standard include file directory **c68k$library:**.

See the /ipath option for information on include file names enclosed in quotes (" ").

Return from Interrupt with RTS

/ireturn=exception (default)

/ireturn=subroutine With the /ireturn=exception option, a function declared as an interrupt procedure will return to the calling function with an RTE instruction.

With the /ireturn=subroutine option, interrupt procedures will return with an RTS instruction.

Write List File to Standard Output

/list[=*listfilename*] (batch default)
/nolist (interactive default)

With the /list[=*listfilename*] option, a listing file containing C source code and any diagnostic messages is created. The default name of the listing file is the source file name with the suffix .lis replacing .c. To override the default, specify a file name after /list.

With the /nolist option, a listing file is not created and diagnostic messages are displayed on the terminal.

The /list=nl option is available to send diagnostic messages to the null device.

Enable Microtec Research Extensions

/mri_extensions (default)
/nomri_extensions

With the /mri_extensions option, Microtec Research extensions to the C language are enabled (e.g., packed structures, assembler inlining, interrupt procedures, and inlining of several library functions). The preprocessor symbol _MRI_EXTENSIONS will be predefined.

The strcpy, memcpy, and strlen functions will be expanded in-line rather than called. If 68881 code is being generated (/fpu), the following function calls will be replaced by in-line 68881 code:

acos	cosh	sin
asin	exp	sinh
atan	fabs	sqrt
atanh	log	tan
cos	log10	tanh

With the /nomri_extensions option, assembler inlining and the keywords interrupt, packed, and unpacked are disabled. Also, the functions mentioned above will not be expanded in-line, but will remain as function calls.

Optimize Code

/optimize=combine_pops (default)
/optimize=nocombine_pops

With the /optimize=combine_pops option, the stack will not be popped after each function call. It will be allowed to grow and will be popped after several function calls. This optimization is known as combining stack pops.

With the /optimize=nocombine_pops option, the compiler generates code to pop the stack after every function call.

/optimize=globalflow
/optimize=noglobalflow (default)

With the /optimize=globalflow option, the global flow optimizer is invoked. If /optimize=globalflow is specified, the /optimize=time or /optimize=size option may be used to control whether optimization favors execution time or code size.

/optimize=inline
/optimize=noinline (default)

The /optimize=inline option must be used in conjunction with the /optimize=globalflow option.

Inlining is a code generation technique whereby the code of a called function is moved into the caller. This eliminates the procedure call overhead (branching and stack allocation) and allows for the possibility of improved code optimization such as constant folding and loop optimizations.

With the /optimize=inline option, the compiler prescans a module looking for functions (usually small functions with only simple local variables) where inlining can enhance performance without costing too much code space. When the module is compiled, the code for these functions will be made inline.

/optimize=local (default)
/optimize=nolocal

With the /optimize=local option, local optimizations are performed.

With the /optimize=nolocal option, code hoisting and cross jump optimizations are disabled.

The /optimize=nolocal option cannot be used in conjunction with the /optimize=globalflow option.

/optimize=register (default)
/optimize=noregister

The /optimize=noregister option works only with the option /optimize=noglobalflow.

If the global optimizer is used (/optimize=globalflow), local variables will be allocated to registers using a register coloring algorithm. If the global optimizer is not used (the default /optimize=noglobalflow), heavily used local variables will be allocated to registers (/optimize=register). The /optimize=noregister option may be used to disable the allocation of heavily used variables to registers.

/optimize=reorder
/optimize=noreorder (default)

With the /optimize=reorder option, the instruction scheduling optimization is enabled.

/optimize=singleret
/optimize=nosingleret (default)

With the /optimize=singleret option, the compiler generates only one exit (return) from a function.

/optimize=size

With the /optimize=size option, the compiler will optimize in favor of code size rather than execution time.

/optimize=stablemem
/optimize=nostablemem (default)

If the global optimizer is used (/optimize=globalflow) and the /optimize=stablemem option is specified, the compiler performs optimizations on global variables. No optimizations will be performed on variables declared as volatile.

The optimizations performed assume that global variables can only be changed by an "unaliased" reference to that variable. A variable is considered to be "aliased" if its address is assigned to a pointer variable.

For a loop-invariant expression containing a global variable, the optimizer will move the expression out of the loop even if the loop contains a pointer dereference on the left hand side of an assignment statement.

With the /optimize=nostablemem option, the compiler may still perform optimizations on a non-volatile variable as long as there is no pointer dereference on the left hand side of the assignment statement within the loop in question. Note that /optimize=nostablemem does not have the same effect as declaring a variable to be volatile.

Warning

The /optimize=stablemem option should not be used on programs that use global variables as operating system memory locations or I/O device registers.

/optimize=time

With the /optimize=time option, the compiler will optimize in favor of execution time rather than code size.

Specify Format of Output Files

/nopage — With the /nopage option, no page breaks or header is produced in the listing file.

/pagelength=*number* — With the /pagelength=*number* option, you can set the page length of the listing file to *number*. The default page length is 55 lines. If the page length specified is less than 10, it will be set to 55 lines.

/show — With the /show option, you can specify an additional item to be displayed in the listing file or the output assembly file.

Suboptions:
=include
=noinclude (default)

With the /show=include option, the contents of each #include file are expanded in the listing file and the assembler source file.

=source
=nosource (default)

With the /show=source option, assembly language and C source code are intermixed as comments in the output assembly file.

/title=*string* With the /title=*string* option, you can specify the title (i.e., *string*) that appears at the top of each page of the listing file. You must specify the option /list[=*listfilename*] to generate a listing file. Enclose the *string* in double quotes if the *string* contains any spaces or punctuation.

Execute C Preprocessor Only

/pp
/nopp (default) With the /pp option, the compiler executes the preprocessor only. Preprocessor output is sent to a file having the same name as the source file name but with the suffix .I. This is useful for debugging complicated macro definitions and deeply nested conditional directives.

With the /nopp option, compilation continues after preprocessing; the preprocessed source file is not saved.

Refer to the *MCC68K Reference Manual* for more information about the /pp option and preprocessor directives.

Suboptions:
=comment
=nocomment (default)
With the /pp=comment option, C comments are displayed in the preprocessor output.

With the /pp=nocomment option, comments are excluded.

Modify Naming Conventions

/noprepend With the /noprepend option, the name modifier (dot or underscore) is suppressed.

/prepend=dot With the /prepend=dot option, a dot is prepended to all symbol names.

/prepend=underscore (default)
With the /prepend=underscore option, an underscore is prepended to all symbol names.

/truncate_identifiers=*number*
/notruncate_identifiers (default)
For a detailed description, see the section *Truncate Name.*

Note

The options /noprepend, /prepend=dot, and /prepend=underscore are mutually exclusive.

The compiler's default naming convention is to prepend an underscore. The MCC68K run-time libraries are built using the default naming convention. Therefore, if a module containing a reference to an item in the library is compiled with a non-default naming convention, that item will be unresolved at link time.

Stop Compilation Due to Errors

/quit=*number* With the /quit=*number* option, compilation will immediately stop if more than *number* errors occur. By default, *number* is 20.

Name the Sections

When using the /rename option, separate multiple renamings by commas. For example:

```
/rename=(code=mycode, const=myconst)
```

/rename=(code=*name*)
With the /rename=(code=*name*), you can set the code section name to *name*. By default, the code section name is code. *name* can be one or two decimal digits, or a symbol beginning with an alphabetic character, ?, ., or _.

/rename=(const=*name*)
With the /rename=(const=*name*), you can set the constant variables section name to *name*. By default, the constant variables section name is const. *name* can be one or two decimal digits, or a symbol beginning with an alphabetic character, ?, ., or _.

/rename=(initvars=*name*)
With the /rename=(initvars=*name*), you can set the initialized data section name to *name*. By default, the initialized data section name is vars. *name* can be one or two decimal digits, or a symbol beginning with an alphabetic character, ?, ., or _.

/rename=(literals=*name*)

With the /rename=(literals=*name*), you can set the compiler-generated literals section name to *name*. By default, the compiler-generated literals section name is literals. *name* can be one or two decimal digits, or a symbol beginning with an alphabetic character, ?, ., or _.

/rename=(module=*name*)

With the /rename=(module=*name*), you can set the module name to *name*. By default, the module name is the source file name stripped of any suffix or prefix. *name* can be any sequence of characters.

/rename=(strings=*name*)

With the /rename=(strings=*name*), you can set the string section name to *name*. By default, the string section name is strings. *name* can be one or two decimal digits, or a symbol beginning with an alphabetic character, ?, ., or _.

/rename=(zerovars=*name*)

With the /rename=(zerovars=*name*), you can set the uninitialized static data section name to *name*. By default, the uninitialized static data name is zerovars. *name* can be one or two decimal digits or a symbol beginning with an alphabetic character, ?, ., or _.

Note

If you compile your C source file with the /weakextern=common option, the linker will allocate all uninitialized global data in the zerovars section. This section can be renamed at link time using the LNK68K alias command. If you compile your C source file with the option /weakextern=public, you can rename the zerovars section at compile time.

Suppress Diagnostic Messages

/nosuppress — With the /nosuppress option, diagnostic messages are not suppressed.

/suppress=error — With the /suppress=error option, you can suppress error, warning, and informational messages.

/suppress=informational (default)

With the /suppress=informational option, you can suppress informational messages.

/suppress=summary With the /suppress=summary option, you can suppress the summary message.

/suppress=warning With the /suppress=warning option, you can suppress warning and informational messages.

Check Program Syntax

/syntax_only
/nosyntax_only (default)

With the /syntax_only option, the compiler checks the program syntax without generating code.

With the /nosyntax_only option, the compiler checks the syntax and generates code.

Generate Trace Information

/trace
/notrace (default) With the /trace option, the compiler generates code to tag each function's entry and exit points. These tags may be used by real-time analysis tools such as the Cadre Soft Analyst to monitor the execution of the program.

For each function, the compiler will allocate two words of data in the tags section. The tag data will be labeled _r_*func*, where *func* is the name of the function. For example:

```
        SECTION tags,,D
        XDEF    _r_main
_r_main:        DCB.B   4,0
```

The compiler will generate the following code in the prologue (function entry):

```
move.w    #10,_r_main
```

The compiler will generate the following code in the epilogue (function exit):

```
move.w    #100,_r_main+2
```

The values `10` and `100` are arbitrary. The analysis tool will detect and record writes to the tag data.

Truncate Name

/truncate_identifiers=*number*
/notruncate_identifiers (default)

With the /truncate_identifiers=*number* option, program identifiers, not language keywords, are truncated to a length of *number*.

With the /notruncate_identifiers option, program identifiers are not truncated.

Undefine a Preprocessor Macro Name

/undefine=*name*

With the /undefine=*name* option, the compiler undefines the defined preprocessor macro specified by *name*. This is equivalent to putting #undef *name* at the beginning of the source file.

Multiple names can be entered by separating them with commas and enclosing them between parentheses. For example:

```
/undefine= (NAME1, NAME2, NAME3)
```

undefines the preprocessor macros `NAME1`, `NAME2`, and `NAME3`.

Treat char as Unsigned

/unsignedchar
/nounsignedchar (default)

With the /unsignedchar option, char variables declared without the signed or unsigned keyword will be considered to be unsigned.

With the /nounsignedchar option, char variables will be considered signed. The _CHAR_SIGNED preprocessor symbol will be predefined.

Initializing Uninitialized Global Data

/weakextern=common (default)

With the /weakextern=common option, weak externals will be output as XCOMs. A "weak external" is a global-level

variable that is not initialized on its declaration. For example:

```
int i;
```

If this variable is not externally defined in another module using **XDEF**, the linker will then allocate it in the **zerovars** section. This section may be renamed at link time using the LNK68K alias command.

/weakextern=initzero With the /weakextern=initzero option, weak externals will be output as XDEFs and initialized to zero. They will be placed in the vars section. This section may be renamed on a per module basis with the /rename=(initvars=*name*) option.

/weakextern=public With the /weakextern=public option, weak externals will be output as XDEFs with the storage defined but not initialized. They will be placed by the compiler in the **zerovars** section. This section may be renamed on a per module basis with the /rename=(zerovars=*name*) option. The zerovars section is set to zero at program start-up time.

Command Line Examples

This section describes MCC68K invocation examples and their results.

Example:

```
C68K program
```

This command compiles the source file `program.c` and produces the assembly file **program.src**.

Example:

```
C68K program/list
```

This command compiles the source file `program.c` and produces two output files: the assembly file **program.src** and the listing file **program.lis**.

Example:

```
C68K /li=list_file iomodule.new/debug
```

This command compiles the source file `iomodule.new`, producing the assembly file **iomodule.src** and the listing file `list_file.lis`. The `/debug` option causes debug information to be generated and included in the assembly file.

Example:

```
C68K condfile /show=source
```

This command compiles `condfile.c` and produces the assembly file **condfile.src**. The `/show` option causes the compiler to intermix C source code with assembly statements in **condfile.src**.

Example:

```
C68K myfile/optimize=time
```

This command compiles the file `myfile.c` and produces the assembly file **myfile.src**. The `/optimize=time` option causes faster code to be generated at the expense of space.

Invoking Other MCC68K Components

The following sections describe the command line syntax for ASM68K and LNK68K — the additional MCC68K components you will need to create an executable program. Refer to the *ASM68K User's Guide* for complete information about using ASM68K and LNK68K.

ASM68K Invocation

Syntax:

`a68k` [*/option*]... *source_filename*[*/option*]...

Description:

`a68k`	The invocation name of the assembler.
/option	An option passed to the assembler to control the generation of the output listing and object module.
source_filename	The assembler input file. The default file name extension is .src. If you enter only the source file name, the assembler automatically generates listing and object module files. The names of the files generated are the same as the source file, but they have the extensions .lis and .obj. Entering only the source file name is a shorthand method of passing file names to the assembler.

LNK68K Invocation

Syntax:

`168k/option[/option]...` *command_filename*

Description:

`168k`	The invocation name of the linker.
`/option`	A qualifier that identifies the named file as a command file. This qualifier must be used with a command file.
/option	An option that controls the generation of the output listing and object module.
command_filename	The name of a file containing one or more linker commands. If a file name extension is not specified, the default is .opt.

The linker creates an absolute object module (default file name extension .abs) and generates a link map file (default file name extension .map). The .map file lists the names of all modules and symbols in the linked files and their absolute addresses. Any errors that may have occurred such as undefined external symbols, duplicate symbols, and segment overlap are listed in the map file as well as on the terminal.

Sample Program

A sample C program, sieve.c, is included on the distribution. The sieve.c source code is shown below. The next section describes how to generate an executable sieve program by compiling sieve.c and then assembling and linking various output files with modules from the run-time library.

```
/* sieve.c -- Eratosthenes Sieve prime number calculation */
/* scaled down with MAX_PRIME set to 17 instead of 8091 */

#define MAX_ITER        1
#define MAX_PRIME       17

char    flags[MAX_PRIME];

main ()
{
        register int i,k;
        int     prime,count,iter;

        for (iter = 1;iter<=MAX_ITER;iter++)
                {
                count = 0;
                for(i = 0; i<MAX_PRIME; i++)
                        flags[i] = 1;
                for(i = 0; i<MAX_PRIME; i++)
                        if(flags[i])
                          {
                          prime = i + i + 3;
                          k = i + prime;
                          while (k < MAX_PRIME)
                               {
                               flags[k] = 0;
                               k += prime;
                               }
                          count++;
                          printf(" prime %d = %d\n", count, prime);
                          }
                }
        printf("\n%d primes\n",count);
}
```

Generating an Executable Program

The MCC68K Compiler allows you to take a C source file and produce an assembler source file. In order to build a program, the assembler source file must be assembled and then linked to produce an executable file. The following sections describe how to do this for the sample program.

1. Compile the sample program by entering:

```
c68k sieve/list/debug
```

This command compiles `sieve.c` and produces the assembly code file sieve.src. The `/list` option produces the listing file sieve.lis.

2. Assemble the compiled program by entering:

```
a68k sieve/nolist
```

The assembler produces the relocatable object modules sieve.obj.

3. The object module created by the assembler must now be linked with the library files and converted into absolute format. Microtec Research provides a sample linker command file called sieve.opt.

Link the object files by entering:

```
l68k sieve/option
```

The linker creates an absolute object module and places it in the file sieve.abs. It also generates a link map and places it in the file sieve.map.

The absolute file sieve.abs is ready to be debugged using XRAY68K or downloaded to an in-circuit emulator or a target system.

> **Note**
>
> You should create linker command files for your applications based on sieve.opt.

Running on a Target System

Since target system environments have different requirements for handling memory organization, shared memory locations, and I/O communication, you will have to write a few custom library routines.

The chapter titled *Embedded Environments* in the *MCC68K Reference Manual* provides helpful information on how to write these customized routines.

Libraries

This section describes the libraries distributed with MCC68K and how they are used.

Distribution Files

Table 2-4 describes the libraries provided with MCC68K and their use.

Table 2-4. Library Use (VMS)

When the Compiler Generates:	Use Library:
/code=abs, /data=abs (default)	mcc68kab.lib
/code=pc, /data=pc	mcc68kpc.lib
/code=pc, /data=a5	mcc68ka5.lib
/code=abs, /data=abs, /cpu=68020, /cpu=68030, /cpu=68040, /cpu=cpu32	mcc68kab020.lib
/code=pc, /data=pc, /cpu=68020, /cpu=68030, /cpu=68040, /cpu=cpu32	mcc68kpc020.lib
/code=pc, /data=a5, /cpu=68020, /cpu=68030, /cpu=68040, /cpu=cpu32	mcc68ka5020.lib

These libraries are collectively referred to as **mcc68k*xx*.lib** in this manual and other MCC68K documentation.

When invoking the linker, you must pass it the correct library based on the **/code**, **/data**, and **/cpu** options that you are using.

The libraries should be used under these conditions:

- Use the **ab** libraries when the compiler is generating absolute data references (**/data=abs**).
- Use the **pc** libraries when the compiler is generating position-independent data references relative to the program counter (**/data=pc**).
- Use the **a5** libraries when the compiler is generating position-independent data references relative to the a5 register (**/data=a5**).
- Use the **020** libraries when the compiler is generating 68020/030/040/CPU32 code (**/cpu=68020**, **/cpu=68030**, **/cpu=68040**, or **/cpu=cpu32**).

The compiler generates absolute code and data by default; therefore, if defaults are used, use the library **mcc68kab.lib**. However, if you specify command line options that produce position-independent data references, you must use the appropriate library. For example, if you compile with the **/data=pc** option, which generates data that is position-independent relative to the program counter, you should link in the library **mcc68kpc.lib**.

If you compile with the **/data=pc** and the **/cpu=68020**, **/cpu=68030**, **/cpu=68040**, or **/cpu=cpu32** options, you should link in the library **mcc68kpc020.lib**. The description of the options located earlier in this chapter indicate which library to use.

Refer to the chapter *Embedded Environments* in the *MCC68K Reference Manual* for more information about position-independent code.

The C Library

The C library (**mcc68kxx.lib**) provides the commonly-used C-language functions. See the *Library Functions* chapter of the *MCC68K Reference Manual* for a description of all library functions provided.

UNIX System Functions

Certain primitive functions cannot be adequately implemented in the C library. These functions are mainly I/O related and must be provided by the operating system. These functions are collectively referred to as the UNIX system functions.

In addition, although the ftell function is not a UNIX system function, it is used by the MCC68K library.

Table 2-5 lists these functions.

Table 2-5. UNIX Functions Used by the MCC68K Library (VMS)

Function	Description
close	Closes a specified file
creat	Creates a new file
_exit	Terminates a program
ftell	Gets the current location in a file
lseek	Sets the current location in a file
open	Opens a specified file
read	Reads bytes from a specified file
sbrk	Allocates memory space
unlink	Unlinks a file name
write	Writes bytes to a specified file

This toolkit is designed to help you develop programs that run in an embedded system, which may not use a standard operating system. If you are developing programs for this kind of environment, you may need to replace some or all of the UNIX system functions in order to successfully run programs in an embedded environment.

Refer to the chapter *Embedded Environments* in the *MCC68K Reference Manual* for more information on providing UNIX system support for the embedded environment.

mcc68kxx.lib contains sample implementations of each UNIX system function. The source code for these sample functions is located in the distribution files.

Start-Up Routine

Refer to the chapter *Embedded Environments* in the *MCC68K Reference Manual* for more information on using a start-up module in the embedded environment.

UNIX / DOS User's Guide 3

Overview

This chapter describes how to use the Microtec Research MCC68K C Compiler on a host computer running the DOS, and UNIX and UNIX-like operating systems.

The chapter includes:

- A description of how to invoke the compiler
- Descriptions of command line options and defaults
- Quick reference tables summarizing the command line options
- Example command lines that make use of various options
- An example of how to compile, assemble, and link a sample program

Invoking the Compilation Driver

The C Compiler is distributed with a UNIX-style driver program that can conveniently invoke the MCC68K C Compiler, ASM68K Assembler, and LNK68K Linker.

This driver program accepts multiple input files, and automatically invokes the appropriate sequence of compiling, assembling, and linking commands to produce an executable file.

This section describes the driver's invocation command, default input and output file name extensions, and default file locations.

Command Line Syntax

The following command invokes the compiler:

`mcc68k` [*-option*... | *source_filename*]...

Description:

`mcc68k`	The name of the compilation driver.
-option	Any of the options described later in the section *Command Line Options*. Each option must be preceded by a dash (-).

Options and *source_filenames* may be specified in any order. If no options are specified, the default options are used.

source_filename The name of one of the following:

- a C program source file
- a 68K assembly language file
- an object file
- an object library

The file name suffixes for *source_filename* are listed in Table 3-1.

Default File Name Extensions

Input and output file name extensions are listed below.

Input File Name Extensions

The driver determines the type of each input file based on its extension. Table 3-1 lists the default extensions for the input files.

Table 3-1. UNIX/DOS Default Input File Name Extensions

File	UNIX Extension	DOS Extension
C intermediate preprocessed file	.i	.i
C source file (from user)	.c	.c
Assembly language file	.s	.src
Relocatable object file	.o	.obj
Object file library	.lib	.lib
Linker command file	.cmd	.cmd

The driver uses the extension to determine how to process each file. It follows these rules:

- If a file's extension is .c or .i, it is compiled with MCC68K.
- If a file's extension is .s or .src, it is assembled with ASM68K.
- If a file has any other extension or no extension, it is assumed to be an object file or a library, and is given to the LNK68K Linker. A warning message is displayed if the file does not have an .o, .obj, or a .lib extension.

Output File Name Extensions

When an output file is created by the compiler, assembler, or linker, it is given a default extension based on its type. These extensions are listed in Table 3-2.

Table 3-2. UNIX/DOS Default Output File Name Extensions

File	UNIX Extension	DOS Extension
Preprocessor output file	.i	.i
Assembly language file	.s	.src
Relocatable object file	.o	.obj
Compiler listing file	.lis	.lis
Executable file	.x	.abs

Input and Output File Locations

If an input file does not have a directory specification, the current default directory is assumed.

If an output file is not specified or if it has no directory specification, the output file is placed in the current default directory.

Environment Variables

The driver and compiler have default directories which are used to find toolkit executables, run-time libraries, standard include files, and to write temporary files. These default directories may be overridden by the environment variables listed in Table 3-3.

Table 3-3. Environment Variables and UNIX/DOS Defaults

Files	UNIX Default	DOS Default	Environment Variable
Executables	/usr/mri/bin	\mcc68k	MRI_68K_BIN
	/usr/mri/bin	\asm68k	MRI_68K_BIN
C standard include files	/usr/mri/include/mcc68k	\mcc68k	MRI_68K_INC
Default command file and libraries	/usr/mri/lib	\mcc68k	MRI_68K_LIB
Temporary files	/tmp	\mcc68k\tmp	MRI_68K_TMP (if defined); otherwise, TMP

Example (UNIX):

If you want to install the distribution files under /usr2/mri/bin, follow these steps:

1. Include /usr2/mri/bin in your current search path.
2. If you are using C-shell, enter these commands:

```
setenv MRI_68K_BIN /usr2/mri/bin
setenv MRI_68K_INC /usr2/mri/include/mcc68k
setenv MRI_68K_LIB /usr2/mri/lib
setenv MRI_68K_TMP /usr2/mri/tmp
```

   If you are using Bourne shell, enter these commands:

```
MRI_68K_BIN=/usr2/mri/bin
MRI_68K_INC=/usr2/mri/include/mcc68k
MRI_68K_LIB=/usr2/mri/lib
MRI_68K_TMP=/usr2/mri/tmp
export MRI_68K_BIN MRI_68K_INC MRI_68K_LIB MRI_68K_TMP
```

   On DOMAIN/OS, substitute the initial "/" with "//".

3. To display the environment variables, enter:

```
printenv
```

Example (DOS):

If you want to install the distribution files in C:\MCC68K, but you want to invoke the compiler while your default drive is D:, follow these steps:

1. Include C:\MCC68K in your current search path.
2. Enter these commands from the terminal:

```
set MRI_68K_BIN=C:\MCC68K;C:\ASM68K
set MRI_68K_INC=C:\MCC68K
set MRI_68K_LIB=C:\MCC68K
set MRI_68K_TMP=C:\MCC68K\TMP
```

3. To display the environment variables, enter:

```
set
```

4. Edit the AUTOEXEC.BAT file to include the commands listed in the previous steps.

On the PC host, the driver will look for the 68000 libraries in \mcc68k\68000 and the 68020/030/040/CPU32 libraries in \mcc68k\68020. The MRI_68K_LIB environment variable may be used to specify an alternate base directory for the libraries. For example, if MRI_68K_LIB is set to \mri68k\lib, the driver will look for the 68000 libraries in \mri68k\lib\68000 and the 68020/030/040/CPU32 libraries in \mri68k\lib\68020.

Considerations for DOS

If MRI_68K_BIN is not defined, then the following directories are searched in the order shown:

1. The drive and directory containing mcc68k.exe.
2. The sibling directory named asm68k if the directory containing mcc68k.exe is named mcc68k.
3. The directory \mcc68k.
4. The directory \asm68k.

If MRI_68K_LIB is not defined, then the following directories are searched in the order shown:

1. The subdirectories 68000 and 68020 below the drive and the directory containing mcc68k.exe.
2. The directories \mcc68k\68000 and \mcc68k\68020.

If **MRI_68K_INC** is not defined, then the following directories are searched in the order shown:

1. The drive and directory containing **mcc68k.exe**.
2. The directory **\mcc68k**.

If **MRI_68K_TMP** is not defined, then the following directories are searched in the order shown:

1. The drive and directory set by the environment variable **TMP**.
2. The drive and directory containing **\mcc68k\tmp**.
3. The current directory.

Compiling Large Modules (DOS only)

The MCC68K ANSI C Compiler reuses memory very effectively. However, it is still possible for the compiler to run out of memory on systems where memory is limited (for example, the PC). The compiler can run out of space because the program has a very large number of symbols. The compiler will issue one of the following messages if it runs out of memory:

```
not enough memory
module is too large for available memory
```

The following sections describe techniques for compiling very large modules. If none of these techniques work for you, you may have to restructure some of your include files or check carefully for the need to include some of them. If you are using very long names, the option -ut*nn* may help alleviate the problem since these names compete for memory space.

Use Extended Memory Compiler

The compiler is capable of taking advantage of extended memory. If you have an extended memory board, use the -B option to invoke the extended memory compiler for compiling very large source modules. The extended memory compiler runs slightly slower than the non-extended memory version.

Preprocess Separately

Macro symbols may be competing for memory with program symbols. Try recompiling the .i file that is produced by the preprocessor:

```
C> mcc68k -P bigfile1.c bigfile2.c
C> mcc68k bigfile1.i bigfile2.i -Og -g
```

Disable ANSI Features to Avoid Prototypes in Standard Include Files

The ANSI branches of the standard include files are full of prototypes which, in turn, are full of parameters. At compile time, each of these parameters requires a "descriptor".

If the -nA option is specified, the parameterless pre-ANSI versions will be selected instead. When the -A option is in effect, the __STDC__ preprocessor symbol is predefined. The standard include files use this symbol to select between prototyped and non-prototyped function declarations.

For example, from **stdio.h**:

```
#if __STDC__
extern int vfprintf(FILE *stream, const char *format,
      va_list arg);
#else
extern int vfprintf();
#endif
```

The standard include file **stdio.h** uses many function prototypes.

Command Line Option Summary

Table 3-4 summarizes the positive form of MCC68K command line options.

Table 3-4. Command Line Option Summary (UNIX/DOS)

Option	Meaning
-A	Sets ANSI-compliant mode.
-acc	References items in the const section according to the -Mc option.
-acd	References items in the const section according to the -Md option.
-aic	References items in the initialized data section according to the -Mc option.
-aid	References items in the initialized data section according to the -Md option.
-alc	References items in the compiler-generated literals section according to the -Mc option.
-ald	References items in the compiler-generated literals section according to the -Md option.
-asc	References items in the strings section according to the -Mc option.
-asd	References items in the strings section according to the -Md option.
-B	Uses extended memory.
-C	Lists comments in the preprocessor output.
-c	Produces an object file, but does not link.
-D*name*[=*string*]	Defines a preprocessor symbol to have the value *string*.
-d*filename*	Reads options from a file.
-E	Invokes C preprocessor only.
-e*command_file*	Passes command file to the linker.

(cont.)

Table 3-4. Command Line Option Summary (cont.)

Option	Meaning
-Fli	Shows the contents of each #include file in the listing file.
-Flp0	Does not print a page header in the listing file.
-Flp*number*	Sets the page length of the listing file to *number*.
-Flt*string*	Specifies title for the listing file.
-Fsi	Shows the contents of each #include file in the assembler source file.
-Fsm	Intermixes assembly language and source code in the output file.
-f	Generates code that uses instructions provided by the floating-point coprocessor.
-Gf	Generates fully qualified path names for input files.
-Gl	Generates line number labels.
-Gm	Generates debug information for multi-statement lines.
-Gr	Generates restricted debug information.
-g	Generates debug information.
-h	Generates code for HP 64000 Series Development Systems.
-I*dir*	Searches the specified path to locate user #include files.
-J*dir*	Searches the specified path to locate standard #include files.
-Kc	Directs the compiler to use the CLR instruction.
-Kf	Generates LINK/UNLK instructions in the function prologue and epilogue.

(cont.)

Table 3-4. Command Line Option Summary (cont.)

Option	Meaning
-Kr	Uses RTS to return from an interrupt function.
-Kt	Generates code to tag a function's exit and entry points.
-Ku	Treats plain chars as unsigned.
-l	Generates a source listing with errors.
-Mca	Directs the compiler to use absolute addressing for all code references.
-Mcp	Directs the compiler to use PC-relative addressing for all code references.
-Mda	Directs the compiler to use absolute addressing for all data references.
-Md*n*	Directs the compiler to use a*n*-relative addressing for all static data references.
-Mdp	Directs the compiler to use PC-relative addressing for all static data references.
-Ml	(68020/030/040/CPU32 modes) Directs the compiler to use 32-bit displacement for a*n*- or PC-relative addressing.
-Mr*n*	Directs the compiler to reserve an a*n* register.
-NC*name*	Sets the const section name to *name*.
-NI*name*	Sets the initialized data section name to *name*.
-NL*name*	Sets the compiler-generated literals section name to *name*.
-NM*name*	Sets the module name to *name*.
-NS*name*	Sets the string section name to *name*.
-NT*name*	Sets the code section name to *name*.

(cont.)

Table 3-4. Command Line Option Summary (cont.)

Option	Meaning
-NZ*name*	Sets the uninitialized static data section name to *name*.
-O	Enables standard optimizations.
-Ob	Optimizes global variables if -Og is also specified.
-Oc	Does not pop the stack after each function call.
-Oe	Generates only one exit (return) from a function.
-Og	Invokes global flow optimizer.
-Oi	Enables inlining.
-Ol	Enables local optimizations.
-OR	Allocates heavily used variables to registers.
-Or	Enables instruction reordering optimizations.
-Os	Optimizes in favor of code size rather than execution time.
-Ot	Optimizes in favor of execution time rather than code size.
-o*filename*	Names the output file.
-P	Creates preprocessor output but does not compile. It also creates a .i file.
-p*processor*	Produces code for a specified processor.
-Q*number*	Stops compilation if more than *number* errors occur.
-Qe	Suppresses error, warning, and informational messages.
-Qi	Suppresses informational messages.
-Qs	Suppresses summary message.
-Qw	Suppresses warning and informational messages.

(cont.)

Table 3-4. Command Line Option Summary (cont.)

Option	Meaning
-S	Generates code for the assembler.
-U*name*	Undefines the defined preprocessor symbol.
-ui*char*	Changes the default insert character in assembler inlining from a back quote (‘) to *char*.
-upd	Prepends a dot to all symbol names.
-upu	Prepends an underscore to all symbol names.
-us	Suppresses name modifier.
-ut*number*	Truncates the length of program identifiers to *number*.
-Vb	Displays the copyright notice and version number. Compilation will continue.
-Vd	Displays commands for invoking components of the toolkit. Commands are not executed.
-Vi	Displays commands for invoking components of the toolkit. Commands are executed.
-Vw	Displays the copyright notice and version number only.
-Wa,*options*	Passes options directly to the assembler.
-Wl,*options*	Passes options directly to the linker.
-Xc	Treats uninitialized global variables as "weak externals" or "C common".
-Xp	Allocates space for global variables that have not been explicitly initialized.
-X0	Allocates space for global variables that have not been explicitly initialized, and initializes them to zero.

(cont.)

Table 3-4. Command Line Option Summary (cont.)

Option	Meaning
-x	Enables Microtec Research extensions to the C language.
-y	Checks source syntax but does not compile.
-Z2	Uses even-alignment for structure members larger than char.
-Z4	Uses quad-alignment for structure members larger than short.

Command Line Options

Command line options specify the names of output files and turn control switches on and off.

Some options have a negative form. The negative form disables or "turns off" the option. You can specify the negative form by entering n before the name of the option. For example:

-n*option*

If conflicting options are specified in the same command line, the last specified option takes effect. For example, if a command line includes two file names specified for the same output file, the second name will be used.

A single letter option without arguments may be followed immediately by another option letter. For example, the combination of options -l -c -g can be written as -lcg.

This page and the following pages describe, in alphabetical order, the command line options for the MCC68K Compiler and its driver on the UNIX and DOS operating systems.

All of the following command line options can be given to the driver program. Some of the options control the behavior of the driver itself. Most of the options, however, do not pertain directly to the driver program, and are simply passed on to the MCC68K Compiler.

Set ANSI-Compliant Mode

-A (default)
-nA

With the -A option, additional features provided by ANSI C are enabled.

With the -nA option, the compiler will accept pre-ANSI C features by disabling any ANSI feature that may invalidate it. For example, if -nA is used, volatile will not be recognized as a keyword.

The -nA option does not guarantee compatibility with earlier versions of the MCC68K Compiler or with other pre-ANSI compilers.

Choose Address Mode for Sections

The compiler and linker allocate code and data to the sections listed in Table 3-5.

Table 3-5. Allocation of Code and Data (UNIX/DOS)

Contents	Section Name
code	code
uninitialized data	zerovars
initialized data	vars
const vars	const
strings	strings
compiler-generated literals	literals
compiler-generated tag data	tags (-Kt option only)

Items in code are referenced according to the address mode chosen with the -Mc option (-Mca or -Mcp). Items in zerovars and tags are referenced according to the address mode chosen with the -Md option (-Mda, -Mdp, or -Md*n*).

You can reference items in the remaining sections (vars, const, strings, literals) according to the address mode chosen with the options -Mc or -Md, depending on where the sections will be placed at link time. If a section is to be placed in ROM (with code), its items should be referenced according to the -Mc option. If a section is to be placed in RAM (with zerovars and vars), its items should be referenced according to the -Md option.

-acc (default)
-acd

With the -acc option, items in const will be referenced according to the -Mc option.

With the -acd option, items in const will be referenced according to the -Md option.

-aic
-aid (default)

With the -aic option, items in vars will be referenced according to the -Mc option.

With the -aid option, items in vars will be referenced according to the -Md option.

-alc (default)
-ald

With the -alc option, items in literals will be referenced according to the -Mc option.

With the -ald option, items in literals will be referenced according to the -Md option.

-asc (default)
-asd

With the -asc option, items in strings will be referenced according to the -Mc option.

With the -asd option, items in strings will be referenced according to the -Md option.

Invoke Extended Memory Compiler (PC only, driver option)

-B
-nB (default)

With the -B option, the compiler takes advantage of extended memory. The extended memory compiler runs slightly slower than the non-extended memory version, but it is able to compile larger modules.

Save Comments

-C
-nC (default)

With the -C option, C comments are displayed in the preprocessor output.

With the -nC option, comments are excluded.

Produce an Object File (driver option)

-c
-nc (default)

With the -c option, the driver produces an object file (.o for UNIX and .obj for DOS), but does not link it to produce an executable file (.x for UNIX and .abs for DOS).

With the -nc option, the driver produces an object file, and the linker produces an executable file.

Define a Preprocessor Macro Name

-D*name*[=*string*]

With the -D*name*[=*string*] option, you can define the value of one or more preprocessor macros. If *string* is not specified, the preprocessor macro has the value 1. This is equivalent to putting #define *name* 1 at the top of the source file.

You can define multiple names by preceding each name with -D. For example:

```
-Dabc -DABC
```

is equivalent to using the preprocessor directives #define abc 1 and #define ABC 1. See the *MCC68K Reference Manual* for more information about -D and preprocessor directives.

Read Options from File (driver option)

-d*filename* With the -d*filename* option, the driver reads command line options from the specified *filename*.

Invoke C Preprocessor Only (driver option)

-E

-nE (default) With the -E option, only the preprocessor is executed. Preprocessor output is sent to standard output. The #line and #pragma directives are not removed, and #include directives are converted to appropriate #line directives so that the original structure of the source files can be determined.

Pass Command File to Linker (driver option)

-e*command_file* With the -e*command_file* option, the driver passes the file specified by *command_file* to the linker. This option also prevents the driver from passing the default library and default linker command file (mcc68k.cmd) to the linker.

Specify Format of Output Files

-Fli

-nFli (default) With the -Fli option, the contents of each #include file are expanded in the listing file. You must specify the -l option to generate a listing file.

-Flp0 With the -Flp0 option, no page breaks or header is produced in the listing file.

-Flp*number* With the -Flp*number* option, you can set the page length of the listing file to *number*. The default page length is 55 lines. If the page length specified is less than 10, it will be set to 55 lines.

-Flt*string*

With the -Flt*string* option, you can specify the title (i.e., *string*) that appears at the top of each page of the listing file. You must specify the -l option to generate a listing file. Enclose the entire *string* in double quotes if the *string* contains any spaces or punctuation.

-Fsi
-nFsi (default)

With the -Fsi option, the contents of each #include file are expanded in the assembler source file. You should also specify the -Fsm and -S options when you specify -Fsi.

-Fsm
-nFsm (default)

With the -Fsm option, assembly language and C source code are intermixed as comments in the output assembly file.

Generate Floating-Point Processor Instructions

-f
-nf (default)

With the -f option, the compiler generates code that utilizes the instructions provided by the 68881 floating-point coprocessor. The preprocessor symbol _FPU will be predefined.

If -p68040 has been specified, the -f option will be ignored.

With the -nf option, all floating-point operations are implemented by making calls to the run-time library.

Produce Debug Information

-Gf
-nGf (default)

The -Gf option should be used in conjunction with the -g option.

With the -Gf option, the compiler generates fully-qualified path names for input files. (A fully-qualified file name has a full path name even if one was not specified for the file name given on the command line.) The fully-qualified file name lets XRAY68K find the source file (.c) even if XRAY68K is not invoked from the directory in which the source file resides.

-Gl
-nGl (default)

The -Gl and the -g option are mutually exclusive.

With the -Gl option, the compiler generates line number labels of the form LL*nn*, where *nn* is the line number. These line number labels will not be recognized by XRAY68K, but may be used in other debugging environments.

-Gm (default)
-nGm

The -Gm option should be used in conjunction with the -g option.

With the -Gm option, the compiler generates debug information for "stepping" on individual statements on lines containing more than one statement.

This option requires the XRAY68K Debugger, version 2.2 or later. If it is used with earlier versions of XRAY68K, it may be necessary to do multiple "step" commands to get past a line of source code containing more than one statement.

-Gr
-nGr (default)

The -Gr option and the -g option are mutually exclusive.

With the -Gr option, the compiler generates "restricted" debug information. For each function, information concerning its frame type, push mask, and start and end addresses will be generated. No debug information will be generated for line numbers or symbols.

This option is typically used when compiling modules that have already been debugged. This option lets the XRAY68K Debugger economically provide call traceback information for the functions in those modules. This functionality is present in XRAY68K, version 2.2 and subsequent versions.

-g
-ng (default)

With the -g option, the compiler generates line number, and variable and tracing information. This information is in Microtec Research format, which can be used by the XRAY68K Debugger. With -g, you must use the Microtec Research ASM68K Assembler and LNK68K Linker.

Support HP 64000

-h
-nh (default)

The -h option should be used in conjunction with the -g option.

With the -h option, the compiler generates code suitable for use on Hewlett-Packard HP 64000 Series Development Systems. The code includes two special labels, R*label* and E*label*, that the HP 64000 uses to identify each function's exit and end point for debugging and timing purposes. R*label* represents the function's return point, and E*label* represents the function's end address. The *label* portion of the symbol name is the name of the function.

The driver also passes the -h option to the assembler and linker.

Add Search Path for Non-Standard Include Files

-I*dir*

With the -I*dir* option, you can specify a search path to be scanned to locate user #include files. If an include file name in your source file is enclosed in double quotes (" "), the compiler will search for the include file:

1. in the directory containing the source file.
2. in the directories specified by the -I option.

   The compiler will search multiple directories if you precede each directory name with -I. For example, if you enter -I*dir1* -I*dir2* -I*dir3*, the compiler searches for #include files in the directory containing the source file, the *dir1* directory, the *dir2* directory, and the *dir3* directory, in that order.
3. in the directories specified with the -J option.
4. in the directories specified by the environment variable MRI_68K_INC, or in the standard include file directory if the environment variable is not defined.

See the -J option for information on include file names enclosed in angle brackets (< >).

Add Search Path for Standard Include Files

-Jdir

With the *-Jdir* option, you can specify a search path to be scanned to locate standard **#include** files. When an include file name in your source file is enclosed in angle brackets (< >), the compiler will search for the include file:

1. in the directories specified by the **-J** option.

   The compiler will search multiple directories if you precede each directory name with **-J**. For example, if you enter *-Jdir1 -Jdir2 -Jdir3*, the compiler searches for standard **#include** files in the *dir1* directory, the *dir2* directory, and the *dir3* directory.

2. in the directories specified by the environment variable **MRI_68K_INC**.

3. in the standard include file directory if the environment variable **MRI_68K_INC** is not defined.

See the **-I** option for information on include file names enclosed in quotes (" ").

Generate Control Prologue/Epilogue Protocol and Code Variations

-Kc (default)
-nKc

With the **-Kc** option, the compiler will use the CLR instruction for assigning zero to register and memory variables.

With the **-nKc** option, the compiler will not use the CLR instruction for assigning zero to non-register variables. On some 68000 processors, the CLR instruction does a read of memory before writing a zero to the location. This may cause problems for memory-mapped I/O ports.

-Kf
-nKf (default)

With the **-Kf** option, all functions will be "framed". A framed function has a LINK/UNLK pair of instructions in the function prologue and epilogue.

With the **-nKf** option, the compiler will generate a frame for a function only if it is needed.

-Kr
-nKr (default)

With the -Kr option, interrupt procedures will return with an RTS instruction.

With the -nKr option, a function declared as an interrupt procedure will return to the calling function with an RTE instruction.

-Kt
-nKt (default)

With the -Kt option, the compiler generates code to tag each function's entry and exit points. These tags may be used by real-time analysis tools such as Cadre Soft Analyst to monitor the execution of the program.

For each function, the compiler will allocate two words of data in the tags section. The tag data will be labeled _r_*func*, where *func* is the name of the function. For example:

```
        SECTION tags,,D
        XDEF    _r_main
_r_main:        DCB.B   4,0
```

The compiler will generate the following code in the prologue (function entry):

```
move.w    #10,_r_main
```

The compiler will generate the following code in the epilogue (function exit):

```
move.w    #100,_r_main+2
```

The values 10 and 100 are arbitrary. The analysis tool will detect and record writes to the tag data.

-Ku
-nKu (default)

With the -Ku option, char variables declared without the signed or unsigned keyword will be considered unsigned.

With the -nKu option, char variables will be considered signed. The preprocessor symbol _CHAR_SIGNED will be predefined.

Write List File to Standard Output

-l
-nl (default) With the -l option, a listing file containing C source code and any diagnostic messages is written to standard output.

With the -nl option, a listing file is not created.

Generate Position-Independent Code and Data

-Mca (default) With the -Mca option, the compiler uses absolute addressing for all code references.

-Mcp With the -Mcp option, the compiler uses the PC-relative addressing mode for all code references. This results in code that is position independent. The preprocessor symbol _PIC will be predefined.

-Mda (default) With the -Mda option, the compiler uses absolute addressing for all data references. If you use this option, link with one of the absolute (ab) libraries.

-Md*n* With the -Md*n* option, the compiler uses a*n*-relative addressing for all references to static data. This results in data that is position independent. The preprocessor symbol _PID will be predefined.

The register number *n* must be in the range 2 to 6. If you use the a5-relative addressing option, link with one of the a5-relative (a5) libraries. For data generated relative to other valid registers, you must build a corresponding library. See the *MCC68K Reference Manual* for information on modifying the default linker command file and for more information about register-relative global data references.

-Mdp With the -Mdp option, the compiler uses PC-relative addressing for all references to static data. This results in data that is position independent. The preprocessor symbol _PID will be predefined.

If you use this option, link with one of the PC-relative (pc) libraries.

-Ml (68020/030/040/CPU32 modes) With the -Ml option, the compiler uses 32-bit displacements for a*n*- or PC-relative addressing.

-Mr*n*

With the -Mr*n* option, the compiler reserves a single a*n* register. The compiler will not generate any code that uses the reserved register. The register number *n* must be in the range 2 to 6.

If you enter -Mr*n* more than once on the command line, the last occurrence of -Mr*n* will override previous occurrences. For example, if you enter:

```
-Mr2 -Mr3
```

the -Mr2 will be discarded and the compiler will not generate any code that uses the a3 register.

Note

The options -Mca and -Mcp are mutually exclusive.

The options -Mda, -Mdp, and -Md*n* are mutually exclusive.

The options -Md*n* and -Mr*n* are mutually exclusive.

Name the Sections

-NC*name*

With the -NC*name* option, you can set the constant variables section name to *name*. By default, the constant variables section name is const. *name* can be one or two decimal digits, or a symbol beginning with an alphabetic character, ?, ., or _.

-NI*name*

With the -NI*name* option, you can set the initialized data section name to *name*. By default, the initialized data section name is vars. *name* can be one or two decimal digits, or a symbol beginning with an alphabetic character, ?, ., or _.

-NL*name*

With the -NL*name* option, you can set the compiler-generated literals section name to *name*. By default, the compiler-generated literals section name is literals. *name* can be one or two decimal digits, or a symbol beginning with an alphabetic character, ?, ., or _.

-NM*name*

With the -NM*name*, you can set the module name to *name*. By default, the module name is the source file name stripped of any suffix or prefix. *name* can be any sequence of characters.

-NS*name*

With the -NS*name* option, you can set the string section name to *name*. By default, the string section name is strings. *name* can be one or two decimal digits, or a symbol beginning with an alphabetic character, ?, ., or _.

-NT*name*

With the -NT*name* option, you can set the code section name to *name*. By default, the code section name is code. *name* can be one or two decimal digits, or a symbol beginning with an alphabetic character, ?, ., or _.

-NZ*name*

With the -NZ*name* option, you can set the uninitialized static data section name to *name*. By default, the uninitialized static data name is zerovars. *name* can be one or two decimal digits, or a symbol beginning with an alphabetic character, ?, ., or _.

Note

If you compile your C source file with the -Xc option, the linker will allocate all uninitialized global data in the zerovars section. This section can be renamed at link time using the LNK68K alias command. If you compile your C source file with the -Xp option, you can rename the zerovars section at compile time.

Optimize Code

-O
-nO (default)

With the -O option, standard optimizations are enabled. The -O option is equivalent to specifying -Ogri.

-Ob
-nOb (default)

If the global optimizer is used (-Og) and the -Ob option is specified, the compiler performs optimizations on global variables. No optimizations will be performed on variables declared as volatile.

The optimizations performed assume that global variables can only be changed by an "unaliased" reference to that variable. A variable is considered to be "aliased" if its address is assigned to a pointer variable.

For a loop-invariant expression containing a global variable, the optimizer will move the expression out of the loop even

if the loop contains a pointer dereference on the left hand side of an assignment statement.

With the -nOb option, the compiler may still perform optimizations on a non-volatile variable as long as there is no pointer dereference on the left hand side of the assignment statement within the loop in question. Note that -nOb does not have the same effect as declaring a variable to be volatile.

> **Warning**
>
> The -Ob option should not be used on programs that use global variables as operating system memory locations or I/O device registers.

-Oc (default)
-nOc

With the -Oc option, the stack will not be popped after each function call. It will be allowed to grow and will be popped after several function calls. This optimization is known as combining stack pops.

With the -nOc option, the compiler generates code to pop the stack after every function call.

-Oe
-nOe (default)

With the -Oe option, the compiler generates only one exit (return) from a function.

-Og
-nOg (default)

With the -Og option, the global flow optimizer is invoked. If -Og is specified, the -Ot or -Os option may be used to control whether optimization favors execution time or code size.

-Oi
-nOi (default)

The -Oi option must be used in conjunction with the -Og option.

Inlining is a code generation technique whereby the code of a called function is moved into the caller. This eliminates the procedure call overhead (branching and stack allocation) and

allows for the possibility of improved code optimization such as constant folding and loop optimizations.

With the -Oi option, the compiler prescans a module looking for functions (usually small functions with only simple local variables) where inlining can enhance performance without costing too much code space. When the module is compiled, the code for these functions will be made inline.

-Ol (default)
-nOl — With the -Ol option, local optimizations are performed.

With the -nOl option, code hoisting and cross jump optimizations are disabled.

The -nOl option cannot be used in conjunction with the -Og option.

-OR (default)
-nOR — The -nOR option works only with -nOg.

If the global optimizer is used (-Og), local variables will be allocated to registers using a register coloring algorithm. If the global optimizer is not used (the default -nOg), heavily used local variables will be allocated to registers (-OR). The -nOR option may be used to disable the allocation of heavily used variables to registers.

-Or
-nOr (default) — With the -Or option, instruction scheduling optimization is enabled.

-Os — With the -Os option, the compiler will optimize in favor of code size rather than execution time.

-Ot — With the -Ot option, the compiler will optimize in favor of execution time rather than code size.

Name the Output File (driver option)

-o*filename* — With the -o*filename* option, the output file is given the name specified by *filename* instead of its default name. The output file can be one of these:

Preprocessed file	If used with the -E or -P options.

Assembly source file	If used with the -S option.
Object file	If used with the -c option.
Executable file	Assumed by default.

Execute C Preprocessor Only (driver option)

-P
-nP (default)

With the -P option, the compiler executes the preprocessor only. Preprocessor output is sent to a file having the same name as the source file name but with the suffix .i. This is useful for debugging complicated macro definitions and deeply nested conditional directives.

With the -nP option, compilation continues after preprocessing; the preprocessed source file is not saved.

Refer to the *MCC68K Reference Manual* for more information about the -P option and preprocessor directives.

Produce Code for Specified Processor

-p*processor*

With the -p*processor* option, the compiler produces code for the specified processor. The possible values for *processor* are **68000** (default), **68010**, **68020**, **68030**, **68040**, and **cpu32**.

If you try to use the **-p68332** option, you will get a warning message indicating that this option is obsolete; the compiler will then default to **-pcpu32**.

The -p*processor* option affects the alignment of global data, local data, and structs. Modules which share structs must be compiled for the same processor. See the *Reference Manual* for more information on alignment.

This option causes one of the following preprocessor symbols to be predefined: **_68000**, **_68010**, **_68020**, **_68030**, **_68040**, **_CPU32**.

If you use the **-p68020**, **-p68030**, **-p68040**, or **-pcpu32** options, link with one of the **020** libraries.

Suppress Diagnostic Messages

-Q*number* With the -Q*number* option, compilation will immediately stop if more than *number* errors occur. By default, *number* is 20.

-Qe With the -Qe option, you can suppress error, warning, and informational messages.

-Qi (default) With the -Qi option, you can suppress informational messages.

-Qs With the -Qs option, you can suppress the summary message.

-Qw With the -Qw option, you can suppress warning and informational messages.

-nQ With the -nQ option, diagnostic messages are not suppressed.

Produce Assembler Source File (driver option)

-S With the -S option, the driver generates assembly language output to be assembled by ASM68K. If -S is used, the driver produces an assembly language file (.s for UNIX and .src for DOS) which can be passed to the assembler to produce an object file (.o for UNIX and .obj for DOS).

Undefine a Preprocessor Macro Name

-U*name* With the -U*name* option, the compiler undefines the defined preprocessor macro specified by *name*. This is equivalent to putting #undef *name* at the beginning of the source file. For example,

```
-UNAME1 -Uname2 -UNAME3
```

undefines the preprocessor macros `NAME1`, `name2`, and `NAME3`.

Modify Naming Conventions

-ui*char* With the -ui*char* option, the default insert character in assembler inlining is changed from a back quote (‘) to *char*.

-upd — With the **-upd** option, a dot is prepended to all symbol names.

-upu (default) — With the **-upu** option, an underscore is prepended to all symbol names.

-us — With the **-us** option, the name modifier (dot or underscore) is suppressed.

-ut0 (default)
-ut*number* — With the **-ut***number* option, program identifiers, not language keywords, are truncated to a length of *number*.

With the **-ut0** option, program identifiers are not truncated.

> **Note**
>
> The options **-upd**, **-upu**, and **-us** are mutually exclusive.

The compiler's default naming convention is to prepend an underscore. The MCC68K run-time libraries are built using the default naming convention. Therefore, if a module containing a reference to an item in the library is compiled with a non-default naming convention, that item will be unresolved at link time.

Control Verbose Output Information

-Vb — With the **-Vb** option, the copyright notice and version number is displayed. Compilation will continue.

-Vd — With the **-Vd** option, the commands that would be used to invoke each component of the toolkit are displayed. The commands are not actually executed.

-Vi — With the **-Vi** option, the commands used to invoke each component of the toolkit are displayed as they are executed.

-Vw — With the **-Vw** option, only the copyright notice and version number are displayed.

-nV (default) — With the **-nV** option, the display of any verbose messages is disabled (equivalent to **-nVb**, **-nVd**, **-nVi**, and **-nVw**).

Pass Options to Tools (driver option)

-Wa,*option1*,*option2*,...

With the -Wa,*option1*,*option2*,... option, the driver passes the specified options directly to the assembler. See the *Microtec Research ASM68K User's Guide* for information about assembler command line options.

-Wl,*option1*,*option2*,...

With the -Wl,*option1*,*option2*,... option, the driver passes the specified options directly to the linker. See the *Microtec Research ASM68K User's Guide* for information about linker command line options.

Initializing Uninitialized Global Data

-Xc (default)

With the -Xc option, weak externals will be output as XCOMs. A "weak external" is a global-level variable that is not initialized on its declaration. For example:

```
int i;
```

If this variable is not externally defined in another module using XDEF, the linker will then allocate it in the zerovars section. This section may be renamed at link time using the LNK68K alias command.

-Xp

With the -Xp option, weak externals will be output as XDEFs with the storage defined but not initialized. They will be placed by the compiler in the zerovars section. This section may be renamed on a per module basis with the -NZ*name* option. The zerovars section is set to zero at program start-up time.

-X0

With the -X0 option, weak externals will be output as XDEFs and initialized to zero. They will be placed in the vars section. This section may be renamed on a per module basis with the -NI*name* option.

Enable Microtec Research Extensions

-x (default)
-nx

With the -x option, Microtec Research extensions to the C language are enabled (e.g., packed structures, assembler inlining, interrupt procedures, and inlining of several

library functions). The _MRI_EXTENSIONS preprocessor symbol will be predefined.

The strcpy, memcpy, and strlen functions will be expanded in-line rather than called. If 68881 code is being generated (-f), the following function calls will be replaced by in-line 68881 code:

acos	cosh	sin
asin	exp	sinh
atan	fabs	sqrt
atanh	log	tan
cos	log10	tanh

With the -nx option, assembler inlining and the keywords interrupt, packed, and unpacked are disabled. Also, the functions mentioned above will not be expanded in-line, but will remain as function calls.

Check Program Syntax (driver option)

-y
-ny (default)

With the -y option, the compiler checks the program syntax without generating code.

With the -ny option, the compiler checks the syntax and generates code.

Choose Structure Alignment

-Z2 (default for -p68000, -p68010, and -pcpu32)
-Z4 (default for -p68020, -p68030, and -p68040)

The default alignment of structure members depends on the processor selected with the -p option. In the 68000/010/CPU32 mode, non-char members will be even-aligned. In 68020/030/040 mode, 4-byte members will be quad-aligned and 2-byte members will be even-aligned.

With the -Z2 option, non-char members will be even-aligned, independent of the processor selected.

With the -Z4 option, 4-byte members will be quad-aligned and 2-byte members will be even-aligned, independent of the processor selected.

The alignment options do not affect packed structures. For information on packed structures, refer to the section titled *Packed Structures* in the *Reference Manual.*

Command Line Examples

This section describes MCC68K invocation examples and their results.

File name extensions in these examples are UNIX-specific and may be different for DOS.

For more information about the libraries described in these examples, see the section titled *Libraries* which is described later in this chapter.

Example:

```
mcc68k program.c
```

This command compiles the source file `program.c` to produce the object file **program.o**. The file **program.o** is then linked with the default linker command file and the library **mcc68kab.lib** (on DOS: **68000\mcc68kab.lib**) to produce the executable file **program.x**. Finally, the object file **program.o** is deleted.

Example:

```
mcc68k module1.c module2.c module3.c -l -Mcp -Md5
```

This command compiles the three source files `module1.c`, `module2.c`, and `module3.c` to produce three object files **module1.o**, **module2.o**, and **module3.o**. A compiler-generated listing of each source file is written to standard output. The generated code will use the PC-relative addressing mode for all code references (`-Mcp`), and the A5-relative addressing mode for all data references (`-Md5`). The three object files are then linked with the default linker command file and the library **mcc68ka5.lib** (on DOS: **68000\mcc68ka5.lib**) to produce the executable file **module1.x**. The name of the executable file is derived from the name of the first source file that appears on the command line; the default extension .x (for UNIX) is added to it. The object files are not deleted in this case; an object file is deleted only if a single source file is successfully linked.

Example:

```
mcc68k -c module.s
```

This command assembles the assembly file `module.s` to produce the object file **module.o**. The `-c` option prevents the compiler from linking this module to produce an executable file.

Example:

```
mcc68k main.c asmfile.s extra.o utilities.lib -p68020
```

This command performs the following:

- `main.c` is compiled to produce **main.o**.
- `asmfile.s` is assembled to produce **asmfile.o**.
- **main.o**, **asmfile.o**, and `extra.o` are linked together along with the default linker command file and the library **mcc68kab020.lib** (on DOS: **68020\mcc68kab.lib**) to produce an executable file named **main.x**. Modules from `utilities.lib` are also linked in if necessary.

Example:

```
mcc68k -Wa,-l>main.l main.c
```

This command compiles `main.c` to produce **main.o**. The `-Wa` option causes the compiler to pass the following `-l` option to the assembler, which generates a listing file named `main.l`.

Example:

```
mcc68k -Wl,-cmain.cmd,-m>main.map main.c
```

This command compiles `main.c` to produce **main.o**. The `-Wl` option causes the compiler to pass the following `-c` option with the `main.cmd` command file to the linker; the linker then generates a map file named `main.map`.

Invoking the Linker

Unless the -c option (see *Produce an Object File*) is entered, the driver automatically invokes the linker. When the linker is invoked, the driver passes it the appropriate library and the default linker command file **mcc68k.cmd**. You can choose an alternate linker command file with the -e option (see *Pass Command File to Linker*). For more information on the LNK68K Linker, see the *Microtec Research ASM68K User's Guide*.

The driver selects which library to pass to the linker based on the -p and -Md options. It is important that the same settings for these options are used on all driver invocations to produce a given executable.

Libraries

This section describes the libraries distributed with MCC68K and how they are used.

Distribution Files

Table 3-6 describes the libraries provided with MCC68K and their use.

Table 3-6. Library Use (UNIX/DOS)

When the Compiler Generates:	Use Library: (UNIX)	Use Library: (DOS)
-Mca, -Mda (default)	mcc68kab.lib	68000\mcc68kab.lib
-Mcp, -Mdp	mcc68kpc.lib	68000\mcc68kpc.lib
-Mcp, -Md5	mcc68ka5.lib	68000\mcc68ka5.lib
-Mca, -Mda, -p68020 -p68030, -p68040, -pcpu32	mcc68kab020.lib	68020\mcc68kab.lib
-Mcp, -Mdp, -p68020 -p68030, -p68040, -pcpu32	mcc68kpc020.lib	68020\mcc68kpc.lib
-Mcp, -Md5, -p68020 -p68030, -p68040, -pcpu32	mcc68ka5020.lib	68020\mcc68ka5.lib

These libraries are collectively referred to as **mcc68k***xx***.lib** in this manual and other MCC68K documentation.

Unless the option in the *Pass Command File to Linker* section in the *User's Guide* is used, the driver will automatically invoke the linker with the appropriate library. However, if you are invoking the linker directly or using an alternate linker command file, you must pass it the correct library.

The libraries should be used under these conditions:

- Use the **ab** libraries when the compiler is generating absolute data references (**-Mda**).
- Use the **pc** libraries when the compiler is generating position-independent data references relative to the program counter (**-Mdp**).
- Use the **a5** libraries when the compiler is generating position-independent data references relative to the A5 register (**-Md5**).
- Use the **020** libraries when the compiler is generating 68020/030/040/CPU32 code (**-p68020**, **-p68030**, **-p68040**, **-pcpu32**).

The compiler generates absolute code and data by default; therefore, if defaults are used, use the library **mcc68kab.lib** on UNIX or **68000\mcc68kab.lib** on DOS. However, if you specify command line options that produce position-independent data references, you must use the appropriate library. For example, if you compile with the **-Mdp** option, which generates data that is position-independent relative to the program counter, you should link in the library **mcc68kpc.lib** on UNIX or **68000\mcc68kpc.lib** on DOS.

If you compile with both the **-Mdp** and the **-p68020, -p68030, -p68040,** or **-pcpu32** options, you should link in the library **mcc68kpc020.lib** on UNIX or, on DOS, **68020\mcc68kpc.lib**. The description of the options located earlier in this chapter indicate which library to use.

Refer to the chapter *Embedded Environments* in the *MCC68K Reference Manual* for more information about position-independent code.

The C Library

The C library (mcc68kxx.lib) provides the commonly used C language functions. See the *Library Functions* chapter of the *MCC68K Reference Manual* for a description of all library functions provided.

UNIX System Functions

Certain primitive functions cannot be adequately implemented in the C library. These functions are mainly I/O related and must be provided by the operating system. These functions are collectively referred to as the UNIX system functions.

In addition, although the ftell function is not a UNIX system function, it is used by the MCC68K library.

Table 3-7 lists these functions.

Table 3-7. UNIX Functions Used by the MCC68K Library (UNIX/DOS)

Function	Description
close	Closes a specified file
creat	Creates a specified file
_exit	Terminates a program
ftell	Gets the current location in a file
lseek	Sets the current location in a file
open	Opens a specified file
read	Reads bytes from a specified file
sbrk	Allocates memory space
unlink	Unlinks a file name
write	Writes bytes to a specified file

This toolkit is designed to help you develop programs that run in an embedded system, which may not use a standard operating system. If you are developing programs for this kind of environment, you may need to replace some or all of the UNIX system functions in order to successfully run programs in an embedded environment.

mcc68kxx.lib contains sample implementations of each UNIX system function. The source code for these sample functions is located in the distribution files.

Start-Up Routine

Refer to the chapter *Embedded Environments* in the *MCC68K Reference Manual* for more information on using a start-up module in the embedded environment.

UNIX / DOS — VMS Equivalences: Appendix A

Table A-1 lists the compiler options on UNIX/DOS hosts and their equivalent VMS options. Option defaults are also shown.

Table A-1. UNIX/DOS — VMS Equivalent Options

UNIX/DOS Options	VMS Options	Notes
-A	/ansi	default
-nA	/noansi	
-acc	/const_addr_as=code	default
-acd	/const_addr_as=data	
-aic	/initvars_addr_as=code	
-aid	/initvars_addr_as=data	default
-alc	/literals_addr_as=code	default
-ald	/literals_addr_as=data	
-asc	/strings_addr_as=code	default
-asd	/strings_addr_as=data	
-B	(not applicable)	
-nB	(not applicable)	default
-C	/pp=comment	
-nC	/pp=nocomment	default
-c	(not applicable)	
-nc	(not applicable)	default
-D*name*[=*string*]	/define=*name*[=*string*]	
-d*filename*	(not applicable)	

(cont.)

Table A-1. UNIX/DOS — VMS Equivalent Options (cont.)

UNIX/DOS Options	VMS Options	Notes
-E	(not applicable)	
-nE	(not applicable)	default
-e*command_file*	(not applicable)	
-Fli	/show=include	
-nFli	/show=noinclude	default
-Flp0	/nopage	
-Flp*number*	/pagelength=*number*	
-Flt*string*	/title=*string*	
-Fsi	/show=include	
-nFsi	/show=noinclude	default
-Fsm	/show=source	
-nFsm	/show=nosource	default
-f	/fpu	
-nf	/nofpu	default
-Gf	/debug=fullfilename	
-nGf	/debug=nofullfilename	default
-Gl	/debug=lines	
-nGl	/debug=nolines	default
-Gm	/debug=multi_stmt	default
-nGm	/debug=nomulti_stmt	
-Gr	/debug=restricted	
-nGr	/debug=norestricted	default
-g	/debug	
-ng	/nodebug	default

(cont.)

Table A-1. UNIX/DOS — VMS Equivalent Options (cont.)

UNIX/DOS Options	VMS Options	Notes
-h	/hp64000	
-nh	/nohp64000	default
-I*dir*	/ipath=*dir*	
-J*dir*	/spath=*dir*	
-Kc	/clr	default
-nKc	/noclr	
-Kf	/frames	
-nKf	/noframes	default
-Kr	/ireturn=subroutine	
-nKr	/ireturn=exception	default
-Kt	/trace	
-nKt	/notrace	default
-Ku	/unsignedchar	
-nKu	/nounsignedchar	default
-l	/list[=*filename*]	batch default
-nl	/nolist	interactive default
-Mca	/code_addresses=absolute	default
-Mcp	/code_addresses=pcrelative	
-Mda	/data_addresses=absolute	default
-Md*n*	/data_addresses=a*n*relative	
-Mdp	/data_addresses=pcrelative	
-Ml	/addresses=long	
	/addresses=short	default
-Mr*n*	/reserve=a*n*	

(cont.)

Table A-1. UNIX/DOS — VMS Equivalent Options (cont.)

UNIX/DOS Options	VMS Options	Notes
-NC*name*	/rename=(const=*name*)	
-NI*name*	/rename=(initvars=*name*)	
-NL*name*	/rename=(literals=*name*)	
-NM*name*	/rename=(module=*name*)	
-NS*name*	/rename=(strings=*name*)	
-NT*name*	/rename=(code=*name*)	
-NZ*name*	/rename=(zerovars=*name*)	
-O	(not applicable)	
-nO	(not applicable)	default
-Ob	/optimize=stablemem	
-nOb	/optimize=nostablemem	default
-Oc	/optimize=combine_pops	default
-nOc	/optimize=nocombine_pops	
-Oe	/optimize=singleret	
-nOe	/optimize=nosingleret	default
-Og	/optimize=globalflow	
-nOg	/optimize=noglobalflow	default
-Oi	/optimize=inline	
-nOi	/optimize=noinline	default
-Ol	/optimize=local	default
-nOl	/optimize=nolocal	
-OR	/optimize=register	default
-nOR	/optimize=noregister	
-Or	/optimize=reorder	

(cont.)

Table A-1. UNIX/DOS — VMS Equivalent Options (cont.)

UNIX/DOS Options	VMS Options	Notes
-nOr	/optimize=noreorder	default
-Os	/optimize=size	
-Ot	/optimize=time	default
-o*filename*	/asm[=*name*]	
-P	/pp	
-nP	/nopp	default
-p68000	/cpu=68000	default
-p68010	/cpu=68010	
-p68020	/cpu=68020	
-p68030	/cpu=68030	
-p68040	/cpu=68040	
-pcpu32	/cpu=cpu32	
-Q*number*	/quit=*number*	
-nQ	/nosuppress	
-Qe	/suppress=error	
-Qi	/suppress=informational	default
-Qs	/suppress=summary	
-Qw	/suppress=warning	
-S	/asm[=*name*]	
-U*name*	/undefine=*name*	
-ui*char*	/insert_char=*char*	
-upd	/prepend=dot	
-upu	/prepend=underscore	default
-us	/noprepend	

(cont.)

Table A-1. UNIX/DOS — VMS Equivalent Options (cont.)

UNIX/DOS Options	VMS Options	Notes
-ut0	/notruncate_identifiers	default
-ut*number*	/truncate_identifiers=*number*	
-Vb	/banner	
	/nobanner	default
-Vd	(not applicable)	
-Vi	(not applicable)	
-Vw	(not applicable)	
-nV	(not applicable)	default
-Wa,*opts*	(not applicable)	
-Wl,*opts*	(not applicable)	
-Xc	/weakextern=common	default
-Xp	/weakextern=public	
-X0	/weakextern=initzero	
-x	/mri_extensions	default
-nx	/nomri_extensions	
-y	/syntax_only	
-ny	/nosyntax_only	default
-Z2	/align=2	(68000/010/CPU32 default)
-Z4	/align=4	(68020/030/040 default)

Index

A

-A option (UNIX/DOS) 3-14
-acc option (UNIX/DOS) 3-15
-acd option (UNIX/DOS) 3-15
acos function 2-19, 3-32
/addresses=long option (VMS) 2-11
/addresses=short option (VMS) 2-11
-aic option (UNIX/DOS) 3-15
-aid option (UNIX/DOS) 3-15
-alc option (UNIX/DOS) 3-16
-ald option (UNIX/DOS) 3-16
/align=2 option (VMS) 2-9
/align=4 option (VMS) 2-9
/ansi option (VMS) 2-10
-asc option (UNIX/DOS) 3-16
-asd option (UNIX/DOS) 3-16
asin function 2-19, 3-32
/asm option (VMS) 2-10
ASM68K Assembler
  description 1-1
Assembler invocation 2-30
atan function 2-19, 3-32
atanh function 2-19, 3-32

B

-B option (UNIX/DOS) 3-6, 3-16
/banner option (VMS) 2-10

C

-C option (UNIX/DOS) 3-16
-c option (UNIX/DOS) 3-16
close function 2-36, 3-38
/clr option (VMS) 2-10
/code_addresses=absolute option (VMS) 2-11
/code_addresses=pcrelative option (VMS) 2-11
Command line
  UNIX/DOS
    examples 3-34
    option summary 3-8
    options 3-14
    syntax 3-1
  VMS
    examples 2-29
    option summary 2-3
    options 2-9
    syntax 2-1
/const_addr_as=code option (VMS) 2-13
/const_addr_as=data option (VMS) 2-13
Conventions, notational vii
cos function 2-19, 3-32
cosh function 2-19, 3-32
/cpu option (VMS) 2-14
creat function 2-36, 3-38

D

-D option (UNIX/DOS) 3-16
-d option (UNIX/DOS) 3-17
Data flow diagram 1-3
/data_addresses=absolute option (VMS) 2-11
/data_addresses=anrelative option (VMS) 2-11
/data_addresses=pcrelative option (VMS) 2-11
/debug option (VMS) 2-14
/debug=fullfilename option (VMS) 2-14
/debug=lines option (VMS) 2-15
/debug=multi_stmt option (VMS) 2-15
/debug=nofullfilename option (VMS) 2-14
/debug=nolines option (VMS) 2-15
/debug=nomulti_stmt option (VMS) 2-15
/debug=norestricted option (VMS) 2-15
/debug=restricted option (VMS) 2-15
/define option (VMS) 2-16
Documentation set
  description v

E

-E option (UNIX/DOS) 3-17

-e option (UNIX/DOS) 3-17
Environment variables 3-3
  UNIX/DOS
    MRI_68K_BIN 3-4
    MRI_68K_INC 3-4
    MRI_68K_LIB 3-4
    MRI_68K_TMP 3-4
    TMP 3-4
Equivalent options A-1
Examples
  command line
    UNIX/DOS 3-34
    VMS 2-29
Executable program
  generation 2-32
  running on a target system 2-33
_exit function 2-36, 3-38
exp function 2-19, 3-32
Extended memory 3-6
Extensions
  file name 2-2, 3-2

F

-f option (UNIX/DOS) 3-18
fabs function 2-19, 3-32
-Fli option (UNIX/DOS) 3-17
-Flp option (UNIX/DOS) 3-17
-Flp0 option (UNIX/DOS) 3-17
-Flt option (UNIX/DOS) 3-18
/fpu option (VMS) 2-16
/frames option (VMS) 2-17
-Fsi option (UNIX/DOS) 3-18
-Fsm option (UNIX/DOS) 3-18
ftell function 2-36, 3-38

G

-g option (UNIX/DOS) 3-19
-Gf option (UNIX/DOS) 3-18
-Gl option (UNIX/DOS) 3-19
-Gm option (UNIX/DOS) 3-19
-Gr option (UNIX/DOS) 3-19

H

-h option (UNIX/DOS) 3-20
/hp64000 option (VMS) 2-17

I

-I option (UNIX/DOS) 3-20
/initvars_addr_as=code option (VMS) 2-13
/initvars_addr_as=data option (VMS) 2-13
Input and output files, location
  UNIX/DOS 3-3
  VMS 2-2
Input file name extensions 3-2
/insert_char option (VMS) 2-17
Invocation
  assembler 2-30
  compiler 2-1, 3-1
  linker 2-31, 3-35
/ipath option (VMS) 2-17
/ireturn=exception option (VMS) 2-18
/ireturn=subroutine option (VMS) 2-18

J

-J option (UNIX/DOS) 3-21

K

-Kc option (UNIX/DOS) 3-21
-Kf option (UNIX/DOS) 3-21
-Kr option (UNIX/DOS) 3-22
-Kt option (UNIX/DOS) 3-22
-Ku option (UNIX/DOS) 3-22

L

-l option (UNIX/DOS) 3-23
LIB68K Librarian
  description 1-2
Libraries 2-34, 3-36
  UNIX system functions 2-35, 3-37
Linker invocation 2-31, 3-35
/list option (VMS) 2-19
/literals_addr_as=code option (VMS) 2-13

/literals_addr_as=data option (VMS) 2-13
LNK68K Linker
  description 1-2
Locations of input and output files
  UNIX/DOS 3-3
  VMS 2-2
log function 2-19, 3-32
log10 function 2-19, 3-32
lseek function 2-36, 3-38

M

-Mca option (UNIX/DOS) 3-23
MCC68K Compiler
  description 1-1
MCC68K compiler package 1-1
mcc68ka5.lib 2-34, 3-36
mcc68ka5020.lib 2-34, 3-36
mcc68kab.lib 2-34, 3-36
mcc68kab020.lib 2-34, 3-36
mcc68kpc.lib 2-34, 3-36
mcc68kpc020.lib 2-34, 3-36
-Mcp option (UNIX/DOS) 3-23
-Md option (UNIX/DOS) 3-23
-Mda option (UNIX/DOS) 3-23
-Mdp option (UNIX/DOS) 3-23
Memory considerations 3-6
  avoiding prototypes 3-7
  preprocessing 3-6
  using extended memory 3-6
Memory, extended 3-6
-Ml option (UNIX/DOS) 3-23
-Mr option (UNIX/DOS) 3-24
MRI_68K_BIN 3-4
MRI_68K_INC 3-4
MRI_68K_LIB 3-4
MRI_68K_TMP 3-4
/mri_extensions option (VMS) 2-19
Multiple statement support 2-15, 3-19

N

-nA option (UNIX/DOS) 3-14
-nB option (UNIX/DOS) 3-16
-NC option (UNIX/DOS) 3-24
-nC option (UNIX/DOS) 3-16
-nc option (UNIX/DOS) 3-16
-nE option (UNIX/DOS) 3-17
-nf option (UNIX/DOS) 3-18
-nFli option (UNIX/DOS) 3-17
-nFsi option (UNIX/DOS) 3-18
-nFsm option (UNIX/DOS) 3-18
-ng option (UNIX/DOS) 3-19
-nGf option (UNIX/DOS) 3-18
-nGl option (UNIX/DOS) 3-19
-nGm option (UNIX/DOS) 3-19
-nGr option (UNIX/DOS) 3-19
-nh option (UNIX/DOS) 3-20
-NI option (UNIX/DOS) 3-24
-nKc option (UNIX/DOS) 3-21
-nKf option (UNIX/DOS) 3-21
-nKr option (UNIX/DOS) 3-22
-nKt option (UNIX/DOS) 3-22
-nKu option (UNIX/DOS) 3-22
-NL option (UNIX/DOS) 3-24
-nl option (UNIX/DOS) 3-23
-NM option (UNIX/DOS) 3-24
-nO option (UNIX/DOS) 3-25
/noansi option (VMS) 2-10
-nOb option (UNIX/DOS) 3-25
/nobanner option (VMS) 2-10
-nOc option (UNIX/DOS) 3-26
/noclr option (VMS) 2-10
/nodebug option (VMS) 2-14
-nOe option (UNIX/DOS) 3-26
/nofpu option (VMS) 2-16
/noframes option (VMS) 2-17
-nOg option (UNIX/DOS) 3-26
/nohp64000 option (VMS) 2-17
-nOi option (UNIX/DOS) 3-26
-nOl option (UNIX/DOS) 3-27
/nolist option (VMS) 2-19
/nomri_extensions (VMS) 2-19
/nooptimize=noglobalflow option (VMS) 2-20
/nopage option (VMS) 2-22
/nopp option (VMS) 2-23
/noprepend option (VMS) 2-23

-nOR option (UNIX/DOS) 3-27
-nOr option (UNIX/DOS) 3-27
/nosuppress option (VMS) 2-25
/nosyntax_only option (VMS) 2-26
Notational conventions vii
/notrace option (VMS) 2-26
/notruncate_identifiers option (VMS) 2-23, 2-27
/nounsignedchar option (VMS) 2-27
-nP option (UNIX/DOS) 3-28
-nQ option (UNIX/DOS) 3-29
-NS option (UNIX/DOS) 3-25
-NT option (UNIX/DOS) 3-25
-nV option (UNIX/DOS) 3-30
-nx option (UNIX/DOS) 3-31
-ny option (UNIX/DOS) 3-32
-NZ option (UNIX/DOS) 3-25

O

-O option (UNIX/DOS) 3-25
-o option (UNIX/DOS) 3-27
-Ob option (UNIX/DOS) 3-25
-Oc option (UNIX/DOS) 3-26
-Oe option (UNIX/DOS) 3-26
-Og option (UNIX/DOS) 3-26
-Oi option (UNIX/DOS) 3-26
-Ol option (UNIX/DOS) 3-27
open function 2-36, 3-38
/optimize=combine_pops option (VMS) 2-20
/optimize=globalflow option (VMS) 2-20
/optimize=inline option (VMS) 2-20
/optimize=local option (VMS) 2-20
/optimize=nocombine_pops option (VMS) 2-20
/optimize=noline option (VMS) 2-20
/optimize=nolocal option (VMS) 2-20
/optimize=noregister option (VMS) 2-21
/optimize=noreorder option (VMS) 2-21
/optimize=nosingleret option (VMS) 2-21
/optimize=nostablemem option (VMS) 2-21
/optimize=register option (VMS) 2-21
/optimize=reorder option (VMS) 2-21
/optimize=singleret option (VMS) 2-21
/optimize=size option (VMS) 2-21
/optimize=stablemem option (VMS) 2-21
/optimize=time option (VMS) 2-22
Options
  negative form of options
    UNIX/DOS 3-14
    VMS 2-9
  positive form of options
    UNIX/DOS 3-14
    VMS 2-9
  specifying conflicting options
    UNIX/DOS 3-14
    VMS 2-9
Options, command line
  summary
    UNIX/DOS 3-8
    VMS 2-3
  UNIX/DOS 3-14
    -A 3-14
    -acc 3-15
    -acd 3-15
    -aic 3-15
    -aid 3-15
    -alc 3-16
    -ald 3-16
    -asc 3-16
    -asd 3-16
    -B 3-16
    -C 3-16
    -c 3-16
    -D 3-16
    -d 3-17
    -E 3-17
    -e 3-17
    -f 3-18
    -Fli 3-17
    -Flp 3-17
    -Flp0 3-17
    -Flt 3-18
    -Fsi 3-18
    -Fsm 3-18
    -g 3-19
    -Gf 3-18

-Gl 3-19
-Gm 3-19
-Gr 3-19
-h 3-20
-I 3-20
-J 3-21
-Kc 3-21
-Kf 3-21
-Kr 3-22
-Kt 3-22
-Ku 3-22
-l 3-23
-Mca 3-23
-Mcp 3-23
-Md 3-23
-Mda 3-23
-Mdp 3-23
-Ml 3-23
-Mr 3-24
-nA 3-14
-nB 3-16
-NC 3-24
-nC 3-16
-nc 3-16
-nE 3-17
-nf 3-18
-nFli 3-17
-nFsi 3-18
-nFsm 3-18
-ng 3-19
-nGf 3-18
-nGl 3-19
-nGm 3-19
-nGr 3-19
-nh 3-20
-NI 3-24
-nKc 3-21
-nKf 3-21
-nKr 3-22
-nKt 3-22
-nKu 3-22
-NL 3-24
-nl 3-23
-NM 3-24
-nO 3-25
-nOb 3-25
-nOc 3-26
-nOe 3-26
-nOg 3-26
-nOi 3-26
-nOl 3-27
-nOR 3-27
-nOr 3-27
-nP 3-28
-nQ 3-29
-NS 3-25
-NT 3-25
-nV 3-30
-nx 3-31
-ny 3-32
-NZ 3-25
-O 3-25
-o 3-27
-Ob 3-25
-Oc 3-26
-Oe 3-26
-Og 3-26
-Oi 3-26
-Ol 3-27
-OR 3-27
-Or 3-27
-Os 3-27
-Ot 3-27
-P 3-28
-p 3-28
-Q 3-29
-Qe 3-29
-Qi 3-29
-Qs 3-29
-Qw 3-29
-S 3-29
-U 3-29
-ui 3-29
-upd 3-30
-upu 3-30
-us 3-30

-ut 3-30
-ut0 3-30
-Vb 3-30
-Vd 3-30
-Vi 3-30
-Vw 3-30
-Wa 3-31
-Wl 3-31
-x 3-31
-X0 3-31
-Xc 3-31
-Xp 3-31
-y 3-32
-Z2 3-32
-Z4 3-32
VMS 2-9
/addresses=long 2-11
/addresses=short 2-11
/align=2 2-9
/align=4 2-9
/ansi 2-10
/asm 2-10
/banner 2-10
/clr 2-10
/code_addresses=absolute 2-11
/code_addresses=pcrelative 2-11
/const_addr_as=code 2-13
/const_addr_as=data 2-13
/cpu 2-14
/data_addresses=absolute 2-11
/data_addresses=anrelative 2-11
/data_addresses=pcrelative 2-11
/debug 2-14
/debug=fullfilename 2-14
/debug=lines 2-15
/debug=multi_stmt 2-15
/debug=nofullfilename 2-14
/debug=nolines 2-15
/debug=nomulti_stmt 2-15
/debug=norestricted 2-15
/debug=restricted 2-15
/define 2-16
/fpu 2-16
/frames 2-17
/hp64000 2-17
/initvars_addr_as=code 2-13
/initvars_addr_as=data 2-13
/insert_char 2-17
/ipath 2-17
/ireturn=exception 2-18
/ireturn=subroutine 2-18
/list 2-19
/literals_addr_as=code 2-13
/literals_addr_as=data 2-13
/mri_extensions 2-19
/noansi 2-10
/nobanner 2-10
/noclr 2-10
/nodebug 2-14
/nofpu 2-16
/noframes 2-17
/nohp64000 2-17
/nolist 2-19
/nomri_extensions 2-19
/nopage 2-22
/nopp 2-23
/noprepend 2-23
/nosuppress 2-25
/nosyntax_only 2-26
/notrace 2-26
/notruncate_identifiers 2-23, 2-27
/nounsignedchar 2-27
/optimize=combine_pops 2-20
/optimize=globalflow 2-20
/optimize=inline 2-20
/optimize=local 2-20
/optimize=nocombine_pops 2-20
/optimize=noglobalflow 2-20
/optimize=noinline 2-20
/optimize=nolocal 2-20
/optimize=noregister 2-21
/optimize=noreorder 2-21
/optimize=nosingleret 2-21
/optimize=nostablemem 2-21
/optimize=register 2-21
/optimize=reorder 2-21

/optimize=singleret 2-21
/optimize=size 2-21
/optimize=stablemem 2-21
/optimize=time 2-22
/pagelength 2-22
/pp 2-23
/pp=comment 2-23
/pp=nocomment 2-23
/prepend=dot 2-23
/prepend=underscore 2-23
/quit 2-24
/rename=(code=name) 2-24
/rename=(const=name) 2-24
/rename=(initvars=name) 2-24
/rename=(literals=name) 2-25
/rename=(module=name) 2-25
/rename=(strings=name) 2-25
/rename=(zerovars=name) 2-25
/reserve=an 2-12
/show 2-22
/show=include 2-22
/show=noinclude 2-22
/show=nosource 2-22
/show=source 2-22
/spath 2-18
/strings_addr_as=code 2-13
/strings_addr_as=data 2-13
/suppress=error 2-25
/suppress=informational 2-26
/suppress=summary 2-26
/suppress=warning 2-26
/syntax_only 2-26
/title 2-23
/trace 2-26
/truncate_identifiers 2-23, 2-27
/undefine 2-27
/unsignedchar 2-27
/weakextern=common 2-27
/weakextern=initzero 2-28
/weakextern=public 2-28
-OR option (UNIX/DOS) 3-27
-Or option (UNIX/DOS) 3-27
Organization of User's Guide v

-Os option (UNIX/DOS) 3-27
-Ot option (UNIX/DOS) 3-27
Output file name extensions 3-3

P

-P option (UNIX/DOS) 3-28
-p option (UNIX/DOS) 3-28
/pagelength option (VMS) 2-22
/pp option (VMS) 2-23
/pp=comment option (VMS) 2-23
/pp=nocomment option (VMS) 2-23
/prepend=dot option (VMS) 2-23
/prepend=underscore option (VMS) 2-23

Q

-Q option (UNIX/DOS) 3-29
-Qe option (UNIX/DOS) 3-29
-Qi option (UNIX/DOS) 3-29
-Qs option (UNIX/DOS) 3-29
Questions vii
/quit option (VMS) 2-24
-Qw option (UNIX/DOS) 3-29

R

read function 2-36, 3-38
Reader's response sheet vii
References vi
/rename=(code=name) option (VMS) 2-24
/rename=(const=name) option (VMS) 2-24
/rename=(initvars=name) option (VMS) 2-24
/rename=(literals=name) option (VMS) 2-25
/rename=(module=name) option (VMS) 2-25
/rename=(strings=name) option (VMS) 2-25
/rename=(zerovars=name) option (VMS) 2-25
/reserve=an option (VMS) 2-12

S

-S option (UNIX/DOS) 3-29
sbrk function 2-36, 3-38
/show option (VMS) 2-22
/show=include option (VMS) 2-22

/show=noinclude option (VMS) 2-22
/show=nosource option (VMS) 2-22
/show=source option (VMS) 2-22
sin function 2-19, 3-32
sinh function 2-19, 3-32
Software performance report vii
/spath option (VMS) 2-18
sqrt function 2-19, 3-32
Start-up routine 2-36, 3-38
/strings_addr_as=code option (VMS) 2-13
/strings_addr_as=data option (VMS) 2-13
Suggestions vii
/suppress=error option (VMS) 2-25
/suppress=informational option (VMS) 2-26
/suppress=summary option (VMS) 2-26
/suppress=warning option (VMS) 2-26
/syntax_only option (VMS) 2-26

T

tan function 2-19, 3-32
tanh function 2-19, 3-32
/title=string (VMS) 2-23
TMP 3-4
/trace option (VMS) 2-26
/truncate_identifiers option (VMS) 2-23, 2-27

U

-U option (UNIX/DOS) 3-29
-ui option (UNIX/DOS) 3-29
/undefine option (VMS) 2-27
UNIX system functions 2-35, 3-37
    _exit 2-36, 3-38
    close 2-36, 3-38
    creat 2-36, 3-38
    ftell 2-36, 3-38
    lseek 2-36, 3-38
    open 2-36, 3-38
    read 2-36, 3-38
    sbrk 2-36, 3-38
    unlink 2-36, 3-38
    write 2-36, 3-38
UNIX/DOS
    command line examples 3-34
    command line option summary 3-8
    command line syntax 3-1
    compiler invocation 3-1
    compiler use 3-1
    file locations 3-3
    libraries provided 3-36
unlink function 2-36, 3-38
/unsignedchar option (VMS) 2-27
-upd option (UNIX/DOS) 3-30
-upu option (UNIX/DOS) 3-30
-us option (UNIX/DOS) 3-30
User's Guide
    organization v
-ut option (UNIX/DOS) 3-30
-ut0 option (UNIX/DOS) 3-30

V

-Vb option (UNIX/DOS) 3-30
-Vd option (UNIX/DOS) 3-30
-Vi option (UNIX/DOS) 3-30
VMS
    command line examples 2-29
    command line option summary 2-3
    command line syntax 2-1
    compiler invocation 2-1
    compiler use 2-1
    file locations 2-2
    libraries provided 2-34
    sample C program 2-32
-Vw option (UNIX/DOS) 3-30

W

-Wa option (UNIX/DOS) 3-31
/weakextern=common option (VMS) 2-27
/weakextern=initzero option (VMS) 2-28
/weakextern=public option (VMS) 2-28
-Wl option (UNIX/DOS) 3-31
write function 2-36, 3-38

X

-x option (UNIX/DOS) 3-31
-X0 option (UNIX/DOS) 3-31
-Xc option (UNIX/DOS) 3-31
-Xp option (UNIX/DOS) 3-31
XRAY68K Debugger
    description 1-2

Y

-y option (UNIX/DOS) 3-32

Z

-Z2 option (UNIX/DOS) 3-32
-Z4 option (UNIX/DOS) 3-32

Reader's Response

Please complete and return the following Reader's Response form to help us improve our documentation. Your comments and suggestions are always appreciated.

Product Name _____
Your Name _____
Company _____
Phone Number _____

Manual Date/Version _____
Title _____
Address _____
_____

What is your level of programming experience?

_____Advanced
_____Intermediate
_____Beginner

In this language?

_____Advanced
_____Intermediate
_____Beginner

Circle the number which best expresses your rating of the information in this manual (*5 is the highest rating*):

Complete?	1	2	3	4	5	Easy to understand?	1	2	3	4	5
Accurate?	1	2	3	4	5	Helpful examples?	1	2	3	4	5
Well organized?	1	2	3	4	5	Helpful procedures?	1	2	3	4	5
Easy to Find?	1	2	3	4	5	At an appropriate level?	1	2	3	4	5
Easy to Read?	1	2	3	4	5	Overall rating?	1	2	3	4	5

3.) Which programming language do you use?

_____

4.) Does the manual provide all the necessary information? If not, what is missing?

_____

5.) Are more examples needed? If so, where?

_____

6.) Did you have any difficulty understanding descriptions or wording? If so, where?

_____

7.) Which sections of this manual are the most important? The most helpful?

_____

8.) What are the best features of this manual?

_____

9.) What are the areas in this manual that need improvement?

_____

10.) Did you find any errors in this manual? (Specify section and page number.)

_____

Software Performance Report Instructions

If problems or errors are encountered when using Microtec Research's software products (including documentation), a Software Performance Report (SPR) should be completed and returned to:

Microtec Research
Software Performance Reports
2350 Mission College Boulevard
Santa Clara, CA 95054

Any comments or suggestions you may have should also be included. All SPRs are directed to our Technical Support Department for analysis and response.

When completing the top section of the SPR, enter the following:

1. Product name and Version number as it appears on the media label or the banner that appears on the screen when the product is invoked.
2. Serial Number as it appears on the media label.
3. Name of the person (Customer) that Microtec Research can contact.
4. Name of the company that the software was Purchased From.
5. Company Name and a shipping Address.
6. Phone Number and/or FAX Number of contact name.
7. Host (machine where the Microtec Research product is run).
8. Host Operating System and Version Number.
9. Report Type showing the type of problem reported.
10. Customer Impact indicating the severity of the problem.

The following guidelines should be followed when completing the bottom section of the SPR:

1. Give as complete a description as possible of the problem, error, or suggestion, including the exact wording of any error messages.
2. If possible, isolate the problem by providing a small example.
3. If the error example is longer than one page of source code, send all information on a tape or floppy disk.
4. Carefully label any media you send and indicate whether you want it returned to you.
5. Include command files, listings, load maps, include files, data files, and all other relevant material with the SPR.
6. If applicable, send sample input and output listings.
7. If the output is from a compiler, send a mixed (source/assembly code) listing.

If the information on the SPR is complete and concise, it will help us give you accurate and timely service to any software problem.

Software Performance Report

Product:	Customer:
	Company Name/Address
Version:	
Serial Number:	
Purchased From:	Phone Number

Host: [ ] Apollo [ ] HP [ ] PC [ ] Other _____
[ ] Sun-3 [ ] Sun-4/SPARC [ ] VMS

Operating System _____ Version _____

Report Type:
- [ ] Problem/Error
- [ ] Documentation
- [ ] Feature Request
- [ ] Other _____

Customer Impact:
- [ ] Heavy
- [ ] Moderate
- [ ] Minor

Problem Description (Attach additional pages if needed):

Problem Recreation (Commands or lines of code used to duplicate problem):

Workaround (Attach additional pages if needed):

Microtec Research Inc.

2350 Mission College Blvd.

Santa Clara, CA 95054

Tel. 408.980.1300

Toll Free 800.950.5554

FAX 408.982.8266